林轩田

《机器学习基石》
《机器学习技法》

精炼笔记



AI 算法工程师必备



林轩田《机器学习基石》课程笔记1 -- The Learning Problem

作者：红色石头　　公众号：AI有道（id：redstonewill）

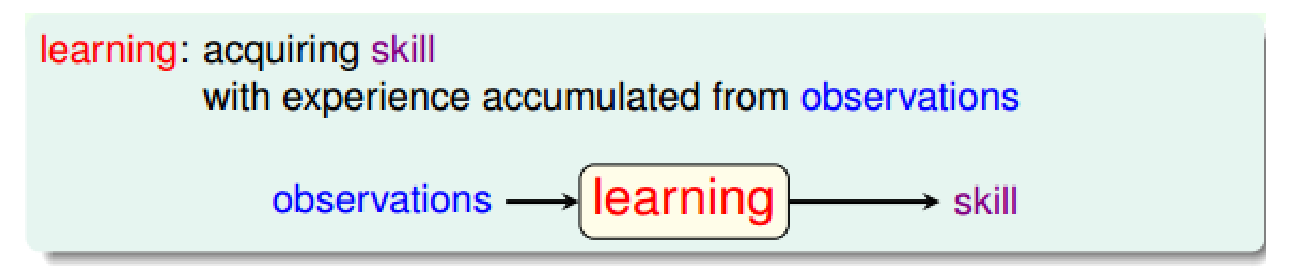
最近在看NTU林轩田的《机器学习基石》课程，个人感觉讲的非常好。整个基石课程分成四个部分：

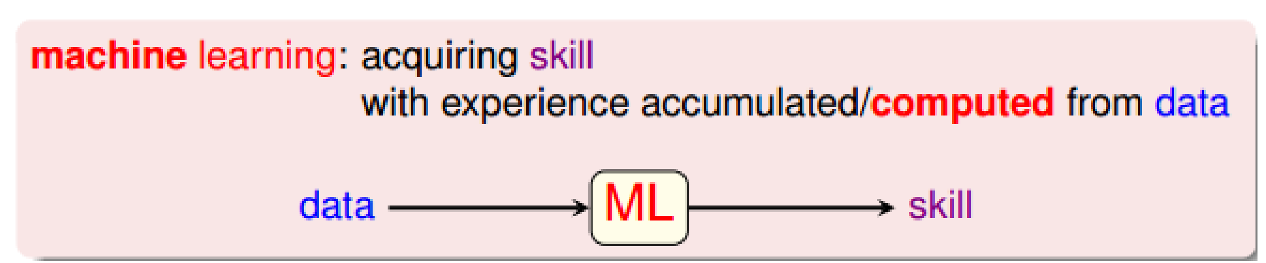
- When Can Machine Learn?
- Why Can Machine Learn?
- How Can Machine Learn?
- How Can Machine Learn Better?

每个部分由四节课组成，总共有16节课。那么，从这篇开始，我们将连续对这门课做课程笔记，共16篇，希望能对正在看这们课的童鞋有所帮助。下面开始第一节课的笔记：The Learning Problem。

一、What is Machine Learning

什么是“学习”？学习就是人类通过观察、积累经验，掌握某项技能或能力。就好像我们从小学习识别字母、认识汉字，就是学习的过程。而机器学习（Machine Learning），顾名思义，就是让机器（计算机）也能向人类一样，通过观察大量的数据和训练，发现事物规律，获得某种分析问题、解决问题的能力。







机器学习可以被定义为：Improving some performance measure with experence computed from data. 也就是机器从数据中总结经验，从数据中找出某种规律或者模型，并用它来解决实际问题。

skill
⇔ improve some performance measure (e.g. prediction accuracy)

machine learning: improving some performance measure
with experience computed from data

data → ML → improved performance measure

什么情况下会使用机器学习来解决问题呢？其实，目前机器学习的应用非常广泛，基本上任何场合都能够看到它的身影。其应用场合大致可归纳为三个条件：

- 事物本身存在某种潜在规律
- 某些问题难以使用普通编程解决
- 有大量的数据样本可供使用

1. exists some ‘underlying pattern’ to be learned
—so ‘performance measure’ can be improved
2. but no programmable (easy) definition
—so ‘ML’ is needed
3. somehow there is data about the pattern
—so ML has some ‘inputs’ to learn from

二、Applications of Machine Learning

机器学习在我们的衣、食、住、行、教育、娱乐等各个方面都有着广泛的应用，我们的生活处处都离不开机器学习。比如，打开购物网站，网站就会给我们自动推荐我们可能会喜欢的商品；电影频道会根据用户的浏览记录和观影记录，向不同用户推荐他们可能喜欢的电影等等，到处都有机器学习的影子。

三、Components of Machine Learning



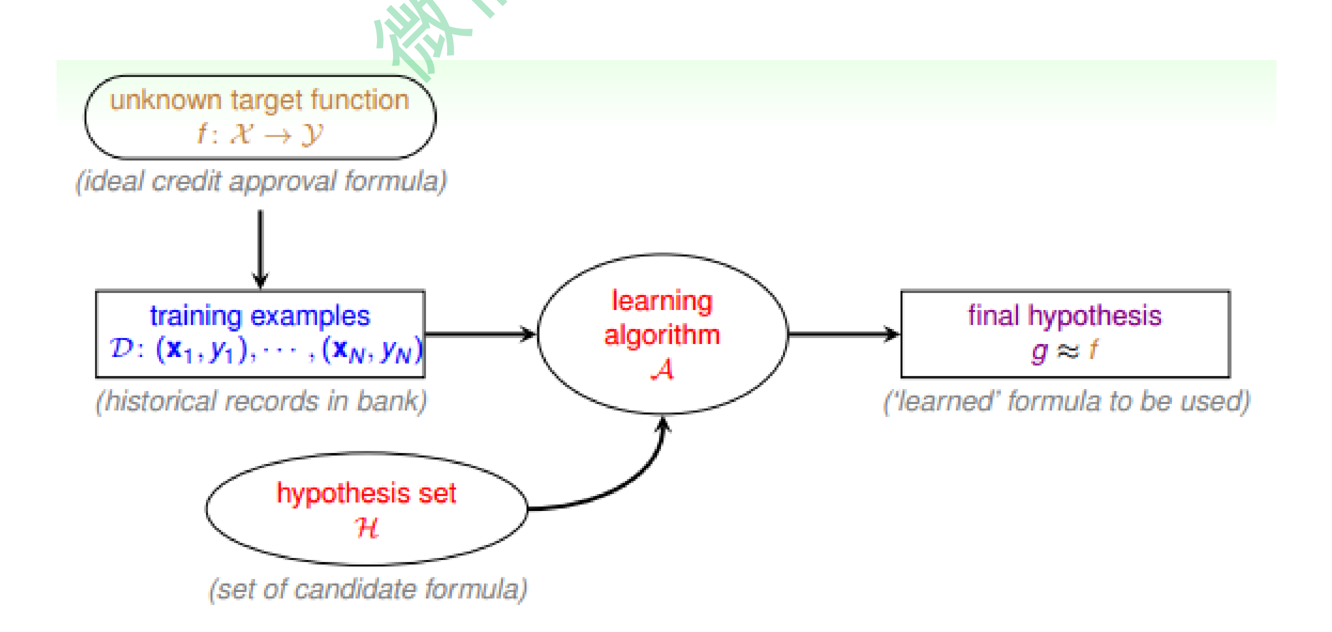
本系列的课程对机器学习问题有一些基本的术语需要注意一下：

- 输入x
- 输出y
- 目标函数f，即最接近实际样本分布的规律
- 训练样本data
- 假设hypothesis，一个机器学习模型对应了很多不同的hypothesis，通过演算法A，选择一个最佳的hypothesis对应的函数称为矩g，g能最好地表示事物的内在规律，也是我们最终想要得到的模型表达式。

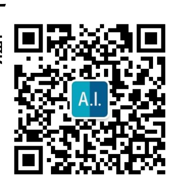
Basic Notations

- input: $\mathbf{x} \in \mathcal{X}$ (customer application)
- output: $y \in \mathcal{Y}$ (good/bad after approving credit card)
- unknown pattern to be learned ⇔ target function:
  $f: \mathcal{X} \to \mathcal{Y}$ (ideal credit approval formula)
- data ⇔ training examples: $\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \cdots, (\mathbf{x}_N, y_N)\}$
  (historical records in bank)
- hypothesis ⇔ skill with hopefully good performance:
  $g: \mathcal{X} \to \mathcal{Y}$ ('learned' formula to be used)

实际中，机器学习的流程图可以表示为：



对于理想的目标函数f，我们是不知道的，我们手上拿到的是一些训练样本D，假设是监督式学习，其中有输入x，也有输出y。机器学习的过程，就是根据先验知识选择模



型，该模型对应的hypothesis set（用H表示），H中包含了许多不同的hypothesis，通过演算法A，在训练样本D上进行训练，选择出一个最好的hypothes，对应的函数表达式g就是我们最终要求的。一般情况下，g能最接近目标函数f，这样，机器学习的整个流程就完成了。

四、Machine Learning and Other Fields

与机器学习相关的领域有：

- 数据挖掘（Data Mining）
- 人工智能（Artificial Intelligence）
- 统计（Statistics）

其实，机器学习与这三个领域是相通的，基本类似，但也不完全一样。机器学习是这三个领域中的有力工具，而同时，这三个领域也是机器学习可以广泛应用的领域，总得来说，他们之间没有十分明确的界线。

五、总结

本节课主要介绍了什么是机器学习，什么样的场合下可以使用机器学习解决问题，然后用流程图的形式展示了机器学习的整个过程，最后把机器学习和数据挖掘、人工智能、统计这三个领域做个比较。本节课的内容主要是概述性的东西，比较简单，所以笔记也相对比较简略。

这里附上林轩田（Hsuan-Tien Lin）关于这门课的主页：
http://www.csie.ntu.edu.tw/~htlin/

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程。



林轩田《机器学习基石》课程笔记2 -- Learning to Answer Yes-No

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课，我们主要简述了机器学习的定义及其重要性，并用流程图的形式介绍了机器学习的整个过程：根据模型H，使用演算法A，在训练样本D上进行训练，得到最好的h，其对应的g就是我们最后需要的机器学习的模型函数，一般g接近于目标函数f。本节课将继续深入探讨机器学习问题，介绍感知机Perceptron模型，并推导课程的第一个机器学习算法：Perceptron Learning Algorithm（PLA）。

一、Perceptron Hypothesis Set

引入这样一个例子：某银行要根据用户的年龄、性别、年收入等情况来判断是否给该用户发信用卡。现在有训练样本D，即之前用户的信息和是否发了信用卡。这是一个典型的机器学习问题，我们要根据D，通过A，在H中选择最好的h，得到g，接近目标函数f，也就是根据先验知识建立是否给用户发信用卡的模型。银行用这个模型对以后用户进行判断：发信用卡（+1），不发信用卡（-1）。

在这个机器学习的整个流程中，有一个部分非常重要：就是模型选择，即Hypothesis Set。选择什么样的模型，很大程度上会影响机器学习的效果和表现。下面介绍一个简单常用的Hypothesis Set：感知机（Perceptron）。

还是刚才银行是否给用户发信用卡的例子，我们把用户的个人信息作为特征向量x，令总共有d个特征，每个特征赋予不同的权重w，表示该特征对输出（是否发信用卡）的影响有多大。那所有特征的加权和的值与一个设定的阈值threshold进行比较：大于这个阈值，输出为+1，即发信用卡；小于这个阈值，输出为-1，即不发信用卡。感知机模型，就是当特征加权和与阈值的差大于或等于0，则输出h(x)=1；当特征加权和与阈值的差小于0，则输出h(x)=-1，而我们的目的就是计算出所有权值w和阈值threshold。



- For $\mathbf{x} = (x_1, x_2, \cdots, x_d)$ 'features of customer', compute a weighted 'score' and

$$\text{approve credit if} \quad \sum_{i=1}^{d} w_i x_i > \text{threshold}$$

$$\text{deny credit if} \quad \sum_{i=1}^{d} w_i x_i < \text{threshold}$$

- $\mathcal{Y}$: $\{+1(\text{good}), -1(\text{bad})\}$, 0 ignored—linear formula $h \in \mathcal{H}$ are

$$h(\mathbf{x}) = \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right)$$

为了计算方便，通常我们将阈值threshold当做w_0，引入一个$x_0 = 1$的量与w_0相乘，这样就把threshold也转变成了权值w_0，简化了计算。h(x)的表达式做如下变换：

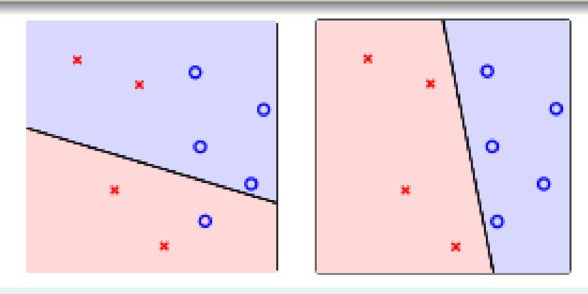
$$\begin{aligned} h(\mathbf{x}) &= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) - \text{threshold}\right) \\ &= \text{sign}\left(\left(\sum_{i=1}^{d} w_i x_i\right) + \underbrace{(-\text{threshold})}_{w_0} \cdot \underbrace{(+1)}_{x_0}\right) \\ &= \text{sign}\left(\sum_{i=0}^{d} w_i x_i\right) \\ &= \text{sign}\left(\mathbf{w}^T \mathbf{x}\right) \end{aligned}$$

为了更清晰地说明感知机模型，我们假设Perceptrons在二维平面上，即$h(x) = sign(w_0 + w_1 x_1 + w_2 x_2)$。其中，$w_0 + w_1 x_1 + w_2 x_2 = 0$是平面上一条分类直线，直线一侧是正类（+1），直线另一侧是负类（-1）。权重w不同，对应于平面上不同的直线。



Perceptrons in $\mathbb{R}^2$

$$h(\mathbf{x}) = \text{sign}\left(w_0 + w_1 x_1 + w_2 x_2\right)$$



- customer features $\mathbf{x}$: points on the plane (or points in $\mathbb{R}^d$)
- labels y: ∘ (+1), × (-1)
- hypothesis h: **lines** (or hyperplanes in $\mathbb{R}^d$)
  —positive on one side of a line, negative on the other side
- different line classifies customers differently

那么，我们所说的Perceptron，在这个模型上就是一条直线，称之为linear(binary) classifiers。注意一下，感知器线性分类不限定在二维空间中，在3D中，线性分类用平面表示，在更高维度中，线性分类用超平面表示，即只要是形如$w^T x$的线性模型就都属于linear(binary) classifiers。

同时，需要注意的是，这里所说的linear(binary) classifiers是用简单的感知器模型建立的，线性分类问题还可以使用logistic regression来解决，后面将会介绍。

二、Perceptron Learning Algorithm(PLA)

根据上一部分的介绍，我们已经知道了hypothesis set由许多条直线构成。接下来，我们的目的就是如何设计一个演算法A，来选择一个最好的直线，能将平面上所有的正类和负类完全分开，也就是找到最好的g，使$g \approx f$。

如何找到这样一条最好的直线呢？我们可以使用逐点修正的思想，首先在平面上随意取一条直线，看看哪些点分类错误。然后开始对第一个错误点就行修正，即变换直线的位置，使这个错误点变成分类正确的点。接着，再对第二个、第三个等所有的错误分类点就行直线纠正，直到所有的点都完全分类正确了，就得到了最好的直线。这种“逐步修正”，就是PLA思想所在。



For $t = 0, 1, \ldots$

1. find a mistake of $\mathbf{w}_t$ called $\left(\mathbf{x}_{n(t)}, y_{n(t)}\right)$

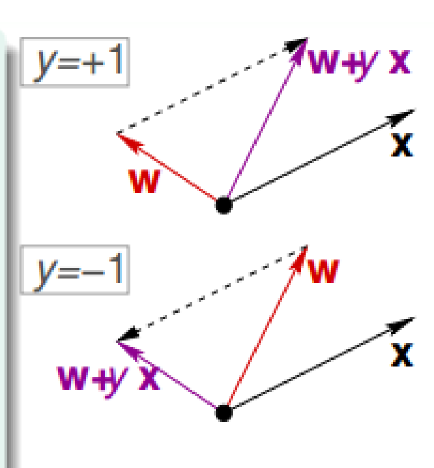
$$\operatorname{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)}$$

2. (try to) correct the mistake by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)} \mathbf{x}_{n(t)}$$

... until no more mistakes

return last $\mathbf{w}$ (called $\mathbf{w}_{\text{PLA}}$) as g



下面介绍一下PLA是怎么做的。首先随机选择一条直线进行分类。然后找到第一个分类错误的点，如果这个点表示正类，被误分为负类，即$w_t^T x_{n(t)} < 0$，那表示w和x夹角大于90度，其中w是直线的法向量。所以，x被误分在直线的下侧（相对于法向量，法向量的方向即为正类所在的一侧），修正的方法就是使w和x夹角小于90度。通常做法是$w \leftarrow w + yx$，$y = 1$，如图右上角所示，一次或多次更新后的$w + yx$与x夹角小于90度，能保证x位于直线的上侧，则对误分为负类的错误点完成了直线修正。

同理，如果是误分为正类的点，即$w_t^T x_{n(t)} > 0$，那表示w和x夹角小于90度，其中w是直线的法向量。所以，x被误分在直线的上侧，修正的方法就是使w和x夹角大于90度。通常做法是$w \leftarrow w + yx$，$y = -1$，如图右下角所示，一次或多次更新后的$w + yx$与x夹角大于90度，能保证x位于直线的下侧，则对误分为正类的错误点也完成了直线修正。

按照这种思想，遇到个错误点就进行修正，不断迭代。要注意一点：每次修正直线，可能使之前分类正确的点变成错误点，这是可能发生的。但是没关系，不断迭代，不断修正，最终会将所有点完全正确分类（PLA前提是线性可分的）。这种做法的思想是“知错能改”，有句话形容它：“A fault confessed is half redressed.”

实际操作中，可以一个点一个点地遍历，发现分类错误的点就进行修正，直到所有点全部分类正确。这种被称为Cyclic PLA。



Cyclic PLA

For $t = 0, 1, \ldots$

1. find **the next** mistake of $\mathbf{w}_t$ called $(\mathbf{x}_{n(t)}, y_{n(t)})$

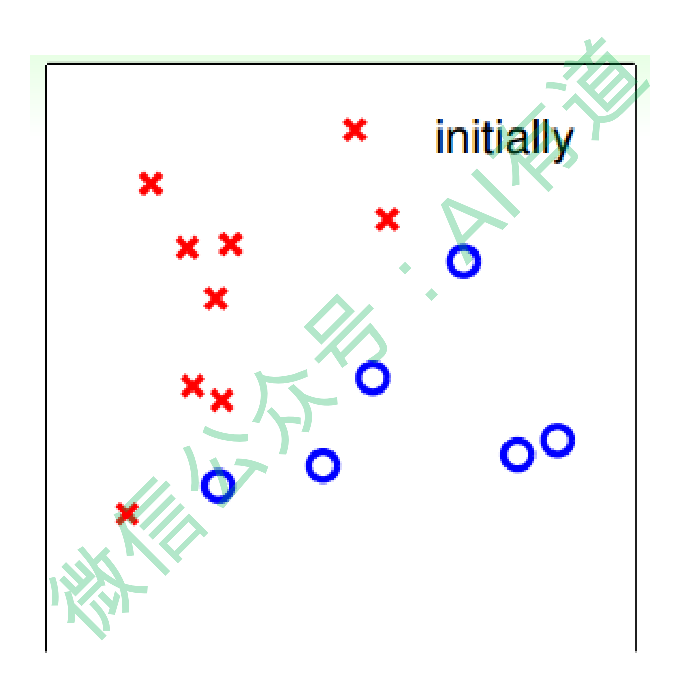
$$\text{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)}$$

2. correct the mistake by

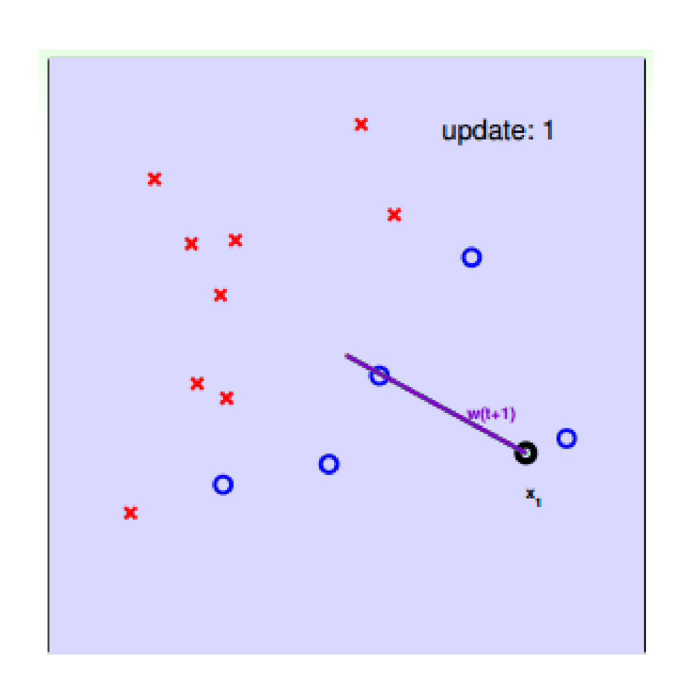
$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)} \mathbf{x}_{n(t)}$$

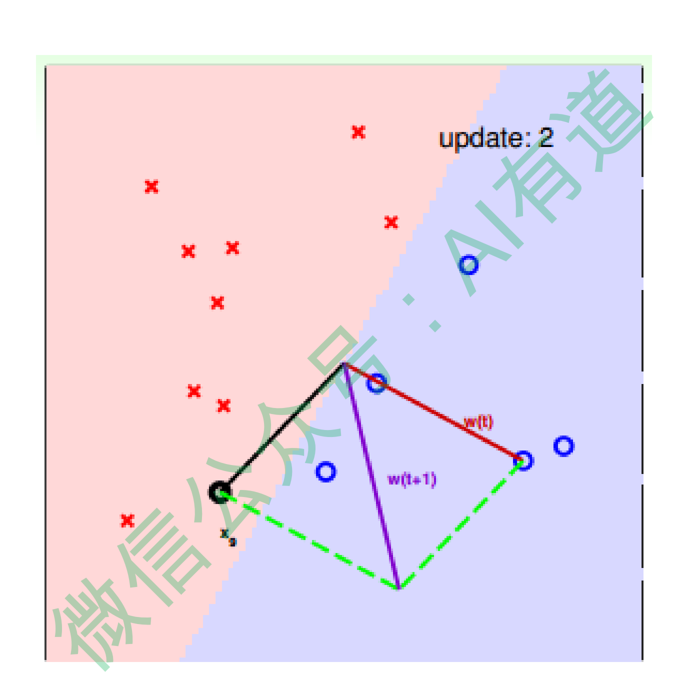
... until **a full cycle of not encountering mistakes**

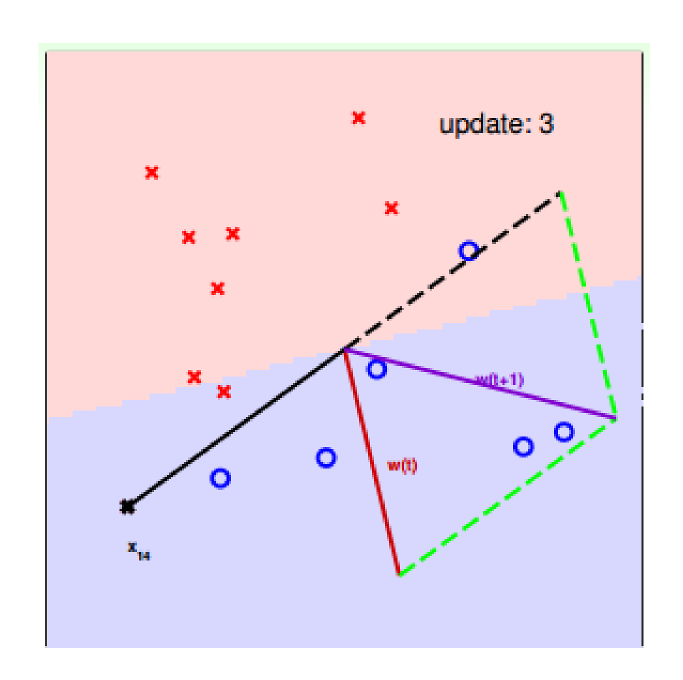
下面用图解的形式来介绍PLA的修正过程：



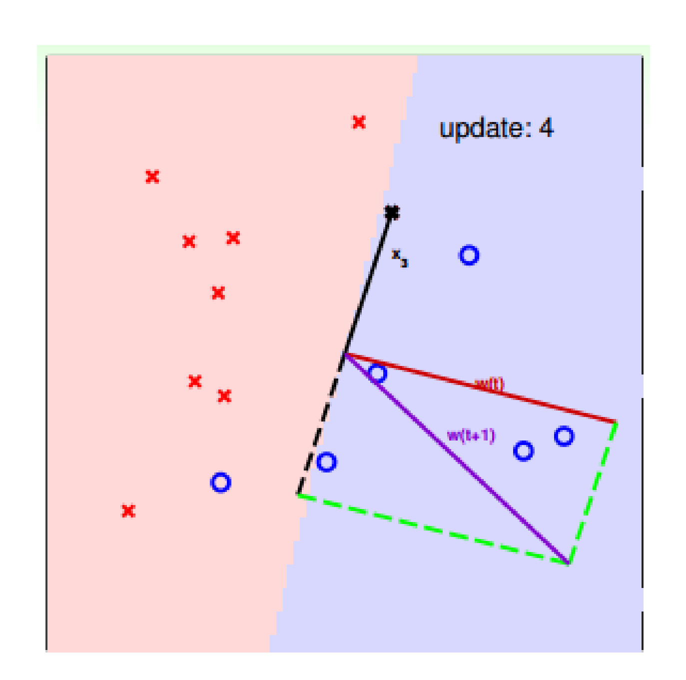


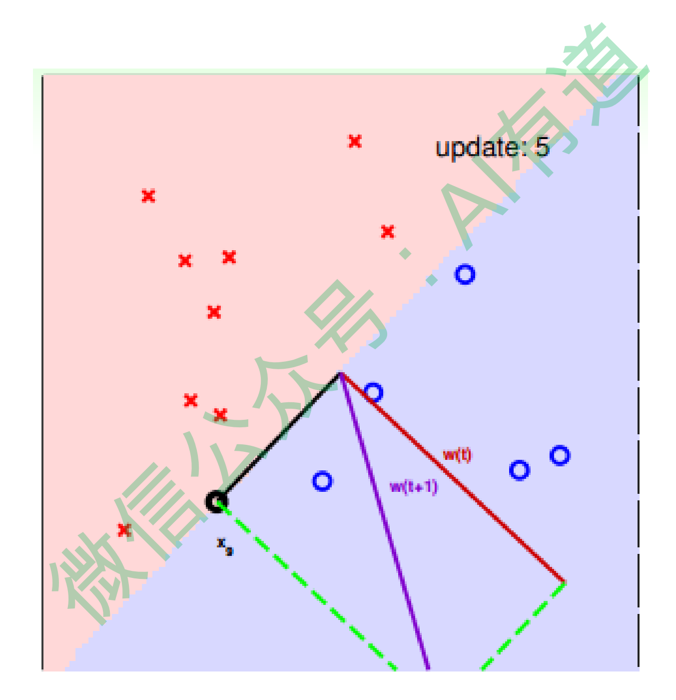

update: 1
w(t+1)
x_1


update: 2
w(t)
w(t+1)
x_9

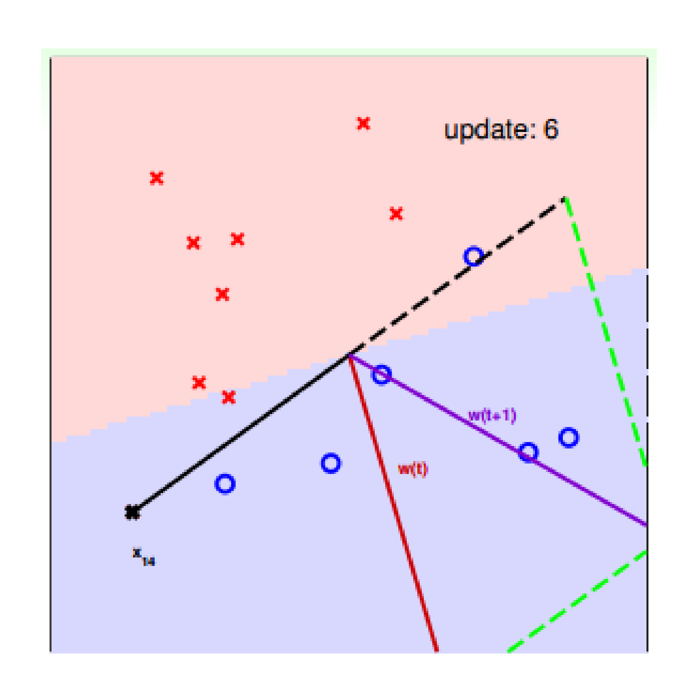

update: 3
w(t+1)
w(t)
x_{14}

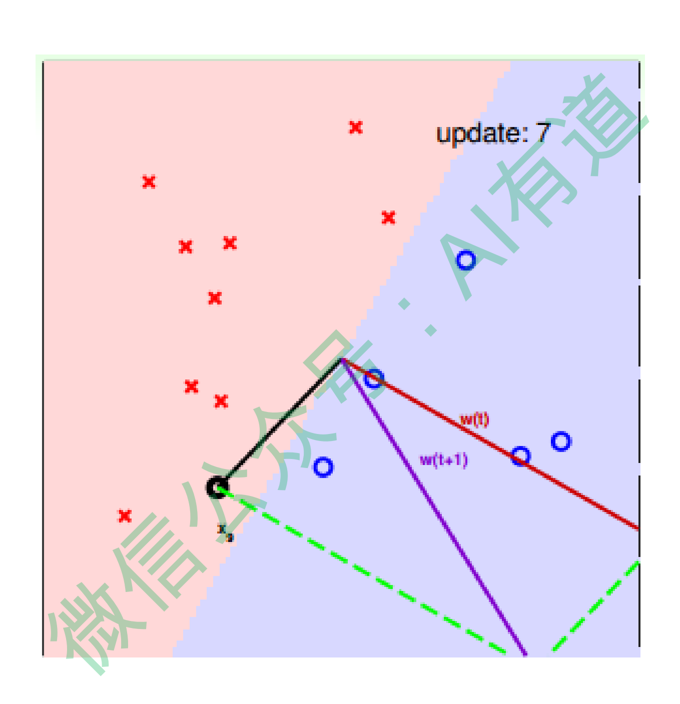


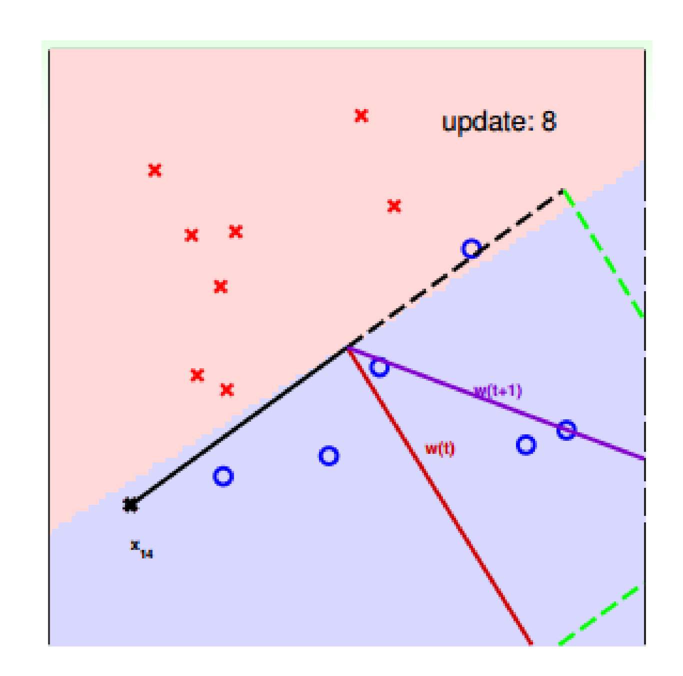

update: 4
w(t)
w(t+1)


update: 5
w(t)
w(t+1)

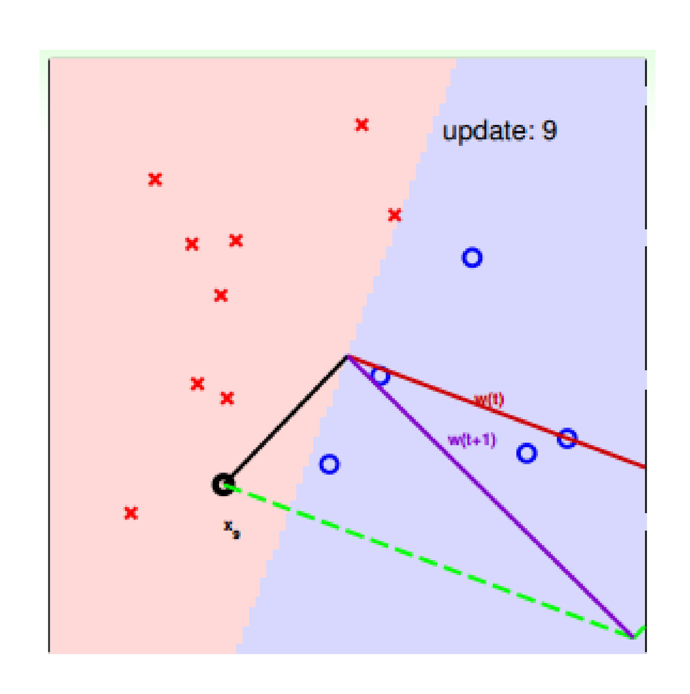


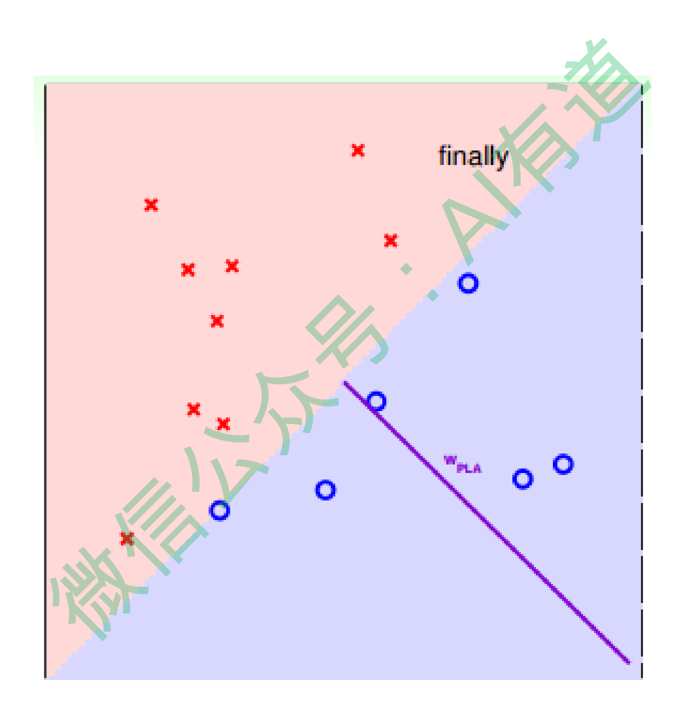

update: 6
w(t+1)
w(t)
x_{14}


update: 7
w(t)
w(t+1)


update: 8
w(t+1)
w(t)
x_{14}







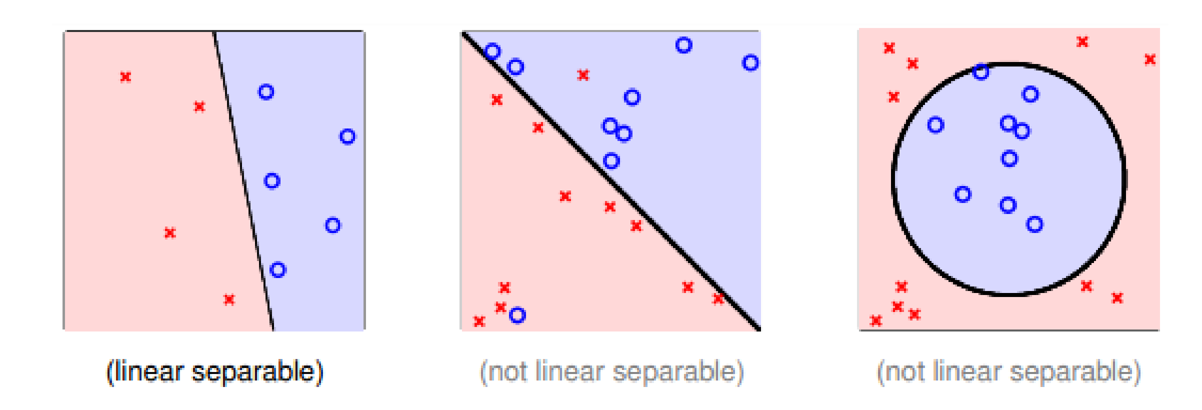
对PLA，我们需要考虑以下两个问题：

- PLA迭代一定会停下来吗？如果线性不可分怎么办？
- PLA停下来的时候，是否能保证$f \approx g$？如果没有停下来，是否有$f \approx g$？

三、Guarantee of PLA

PLA什么时候会停下来呢？根据PLA的定义，当找到一条直线，能将所有平面上的点都分类正确，那么PLA就停止了。要达到这个终止条件，就必须保证D是线性可分（linear separable）。如果是非线性可分的，那么，PLA就不会停止。





对于线性可分的情况，如果有这样一条直线，能够将正类和负类完全分开，令这时候的目标权重为w_f，则对每个点，必然满足$y_n = sign(w_f^T x_n)$，即对任一点：

$$y_{n(t)} \mathbf{w}_f^T \mathbf{x}_{n(t)} \geq \min_n y_n \mathbf{w}_f^T \mathbf{x}_n > 0$$

PLA会对每次错误的点进行修正，更新权重w_{t+1}的值，如果w_{t+1}与w_f越来越接近，数学运算上就是内积越大，那表示w_{t+1}是在接近目标权重w_f，证明PLA是有学习效果的。所以，我们来计算w_{t+1}与w_f的内积：

$$\begin{aligned}
\mathbf{w}_f^T \mathbf{w}_{t+1} &= \mathbf{w}_f^T \left(\mathbf{w}_t + y_{n(t)} \mathbf{x}_{n(t)} \right) \\
&\geq \mathbf{w}_f^T \mathbf{w}_t + \min_n y_n \mathbf{w}_f^T \mathbf{x}_n \\
&> \mathbf{w}_f^T \mathbf{w}_t + 0.
\end{aligned}$$

从推导可以看出，w_{t+1}与w_f的内积跟w_t与w_f的内积相比更大了。似乎说明了w_{t+1}更接近w_f，但是内积更大，可能是向量长度更大了，不一定是向量间角度更小。所以，下一步，我们还需要证明w_{t+1}与w_t向量长度的关系：



$\mathbf{w}_t$ changed only when mistake

$\Leftrightarrow \text{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)} \Leftrightarrow y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)} \leq 0$

- mistake 'limits' $\|\mathbf{w}_t\|^2$ growth, even when updating with 'longest' $\mathbf{x}_n$

$$
\begin{aligned}
\|\mathbf{w}_{t+1}\|^2 &= \|\mathbf{w}_t + y_{n(t)} \mathbf{x}_{n(t)}\|^2 \\
&= \|\mathbf{w}_t\|^2 + 2 y_{n(t)} \mathbf{w}_t^T \mathbf{x}_{n(t)} + \|y_{n(t)} \mathbf{x}_{n(t)}\|^2 \\
&\leq \|\mathbf{w}_t\|^2 + 0 + \|y_{n(t)} \mathbf{x}_{n(t)}\|^2 \\
&\leq \|\mathbf{w}_t\|^2 + \max_n \|y_n \mathbf{x}_n\|^2
\end{aligned}
$$

w_t只会在分类错误的情况下更新，最终得到的$||w_{t+1}^2||$相比$||w_t^2||$的增量值不超过$max||x_n^2||$。也就是说，w_t的增长被限制了，w_{t+1}与w_t向量长度不会差别太大！

如果令初始权值$w_0 = 0$，那么经过T次错误修正后，有如下结论：

$$\frac{w_f^T}{||w_f||} \frac{w_T}{w_T} \geq \sqrt{T} \cdot constant$$

下面贴出来该结论的具体推导过程：



Given 3 conditions
1 w_f perfectly fitting those data means:

$$y_{n(t)} w_f^T x_{n(t)} \geq \min_n y_n w_f^T x_n > 0 \tag{1}$$

2 w_t changed only when making mistakes means:

$$sign(w_t^T x_{n(t)}) \neq y_{n(t)} \Leftrightarrow y_{n(t)} w_t^T x_{n(t)} \leq 0 \tag{2}$$

3 we make corrections by:

$$w_t = w_{t-1} + y_{n(t)} x_{n(t)} \tag{3}$$

We want to prove after t mistake corrections starting from 0, we will get:

$$\frac{w_f^T}{\|w_f\|} \frac{w_t}{\|w_t\|} \geq \sqrt{T} \cdot \text{constant}$$

it means 2 things:
1 Because the above equation is less equal than 1, we also prove the T will have upper boundary, thus the algorithm will stop at some time;
2 the left part the above equation means the angle of w_f and w_t, and their angle is decreasing, in other words w_t is approaching w_f, thus we are on the right way to get the solution
Here is the prove:
the primary principle is to replace all the variables which in above equations are w_t and $\|w_t\|$
for w_t we have:

$$\begin{aligned}
w_f^T w_t &= w_f^T (w_{t-1} + y_{n(t-1)} x_{n(t-1)}) && \text{using (3)} \\
&\geq w_f^T w_{t-1} + \min_n y_n w_f^T x_n && \text{using (1)} \\
&\geq w_0 + T \cdot \min_n y_n w_f^T x_n && \text{applying T times} \\
&\geq T \cdot \min_n y_n w_f^T x_n
\end{aligned}$$

for $\|w_t\|$ we have:

$$\begin{aligned}
\|w_t\|^2 &= \left\|w_{t-1} + y_{n(t-1)} x_{n(t-1)}\right\|^2 && \text{using (3)} \\
&= \|w_{t-1}\|^2 + 2 y_{n(t-1)} w_t^T x_{n(t-1)} + \left\|y_{n(t-1)} x_{n(t-1)}\right\|^2 \\
&\leq \|w_{t-1}\|^2 + 0 + \left\|y_{n(t-1)} x_{n(t-1)}\right\|^2 && \text{using (2)} \\
&\leq \|w_{t-1}\|^2 + \max_n \ \|x_n\|^2 \\
&\leq \|w_0\| + T \cdot \max_n \ \|x_n\|^2 = T \cdot \max_n \ \|x_n\|^2
\end{aligned}$$



applying above 2 conclusions to the left part of the equation we finally get:

$$
\begin{aligned}
\frac{w_f^T}{\|w_f\|}\frac{w_T}{\|w_T\|} &= \frac{T \cdot \underset{n}{min}\ y_n w_f^T x_n}{\left\|w_f^T\right\| \cdot \|w_t\|} \\
&\geq \frac{T \cdot \underset{n}{min}\ y_n w_f^T x_n}{\left\|w_f^T\right\| \cdot \sqrt{T} \cdot \underset{n}{max}\ \|x_n\|} \\
&\geq \frac{\sqrt{T} \cdot \underset{n}{min}\ y_n w_f^T x_n}{\left\|w_f^T\right\| \cdot \underset{n}{max}\ \|x_n\|} = \sqrt{T} \cdot \text{constant}
\end{aligned}
$$

Because the $\frac{w_f^T}{\|w_f\|}\frac{w_T}{\|w_T\|} \leq 1$, we finally have:

$$
\begin{aligned}
&\frac{\sqrt{T} \cdot \underset{n}{min}\ y_n w_f^T x_n}{\left\|w_f^T\right\| \cdot \underset{n}{max}\ \|x_n\|} \leq 1 \\
\Leftrightarrow T \leq &\frac{\underset{n}{max}\ \|x_n\|^2 \cdot \left\|w_f^T\right\|^2}{\underset{n}{min^2}\ y_n w_f^T x_n} = \frac{R^2}{\rho^2}
\end{aligned}
$$

上述不等式左边其实是w_T与w_f夹角的余弦值，随着T增大，该余弦值越来越接近1，即w_T与w_f越来越接近。同时，需要注意的是，$\sqrt{T} \cdot constant \leq 1$，也就是说，迭代次数T是有上界的。根据以上证明，我们最终得到的结论是：w_{t+1}与w_f的是随着迭代次数增加，逐渐接近的。而且，PLA最终会停下来（因为T有上界），实现对线性可分的数据集完全分类。

四、Non-Separable Data

上一部分，我们证明了线性可分的情况下，PLA是可以停下来并正确分类的，但对于非线性可分的情况，w_f实际上并不存在，那么之前的推导并不成立，PLA不一定会停下来。所以，PLA虽然实现简单，但也有缺点：



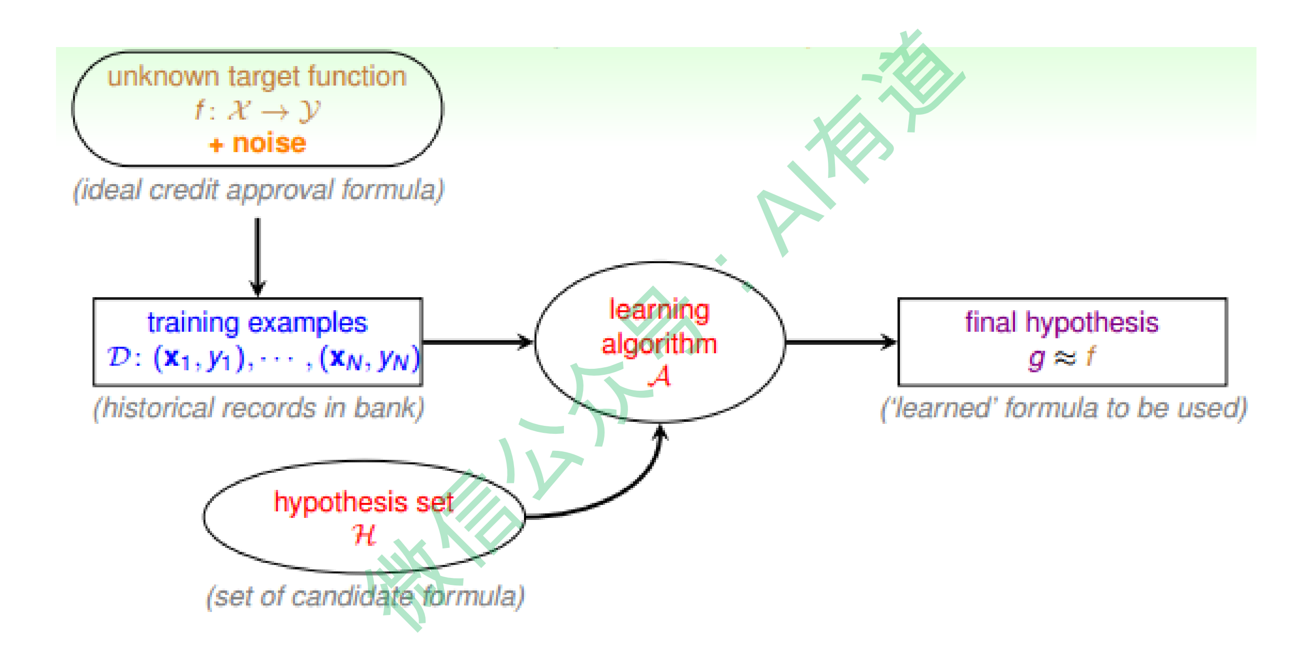
Pros

simple to implement, fast, works in any dimension d

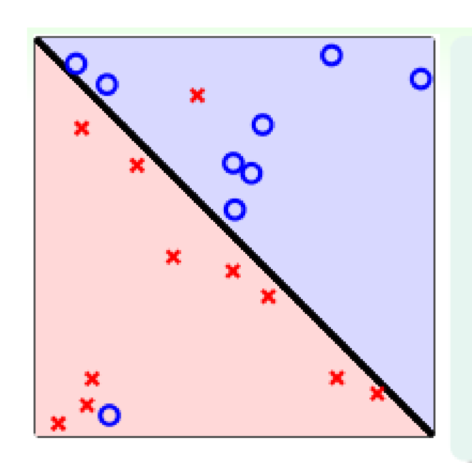
Cons

- **'assumes' linear separable** $\mathcal{D}$ to halt
  —property unknown in advance (no need for PLA if we know $\mathbf{w}_f$)
- not fully sure **how long halting takes** (ρ depends on $\mathbf{w}_f$)
  —though practically fast

对于非线性可分的情况，我们可以把它当成是数据集D中掺杂了一下noise，事实上，大多数情况下我们遇到的D，都或多或少地掺杂了noise。这时，机器学习流程是这样的：



在非线性情况下，我们可以把条件放松，即不苛求每个点都分类正确，而是容忍有错误点，取错误点的个数最少时的权重w：



- assume 'little' noise: $y_n = f(\mathbf{x}_n)$ **usually**
- if so, $g \approx f$ on $\mathcal{D} \Leftrightarrow y_n = g(\mathbf{x}_n)$ **usually**
- how about

$$\mathbf{w}_g \leftarrow \mathop{\mathrm{argmin}}_{\mathbf{w}} \sum_{n=1}^{N} [\![y_n \neq \mathrm{sign}(\mathbf{w}^T \mathbf{x}_n)]\!]$$

—**NP-hard to solve, unfortunately**



事实证明，上面的解是NP-hard问题，难以求解。然而，我们可以对在线性可分类型中表现很好的PLA做个修改，把它应用到非线性可分类型中，获得近似最好的g。

修改后的PLA称为Packet Algorithm。它的算法流程与PLA基本类似，首先初始化权重w_0，计算出在这条初始化的直线中，分类错误点的个数。然后对错误点进行修正，更新w，得到一条新的直线，在计算其对应的分类错误的点的个数，并与之前错误点个数比较，取个数较小的直线作为我们当前选择的分类直线。之后，再经过n次迭代，不断比较当前分类错误点个数与之前最少的错误点个数比较，选择最小的值保存。直到迭代次数完成后，选取个数最少的直线对应的w，即为我们最终想要得到的权重值。

initialize pocket weights $\hat{\mathbf{w}}$

For $t = 0, 1, \cdots$

1. find a (random) mistake of $\mathbf{w}_t$ called $(\mathbf{x}_{n(t)}, y_{n(t)})$
2. (try to) correct the mistake by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)}\mathbf{x}_{n(t)}$$

3. if $\mathbf{w}_{t+1}$ makes fewer mistakes than $\hat{\mathbf{w}}$, replace $\hat{\mathbf{w}}$ by $\mathbf{w}_{t+1}$

...until enough iterations

return $\hat{\mathbf{w}}$ (called $\mathbf{w}_{\text{POCKET}}$) as g

如何判断数据集D是不是线性可分？对于二维数据来说，通常还是通过肉眼观察来判断的。一般情况下，Pocket Algorithm要比PLA速度慢一些。

五、总结

本节课主要介绍了线性感知机模型，以及解决这类感知机分类问题的简单算法：PLA。我们详细证明了对于线性可分问题，PLA可以停下来并实现完全正确分类。对于不是线性可分的问题，可以使用PLA的修正算法Pocket Algorithm来解决。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程。



林轩田《机器学习基石》课程笔记3 -- Types of Learning

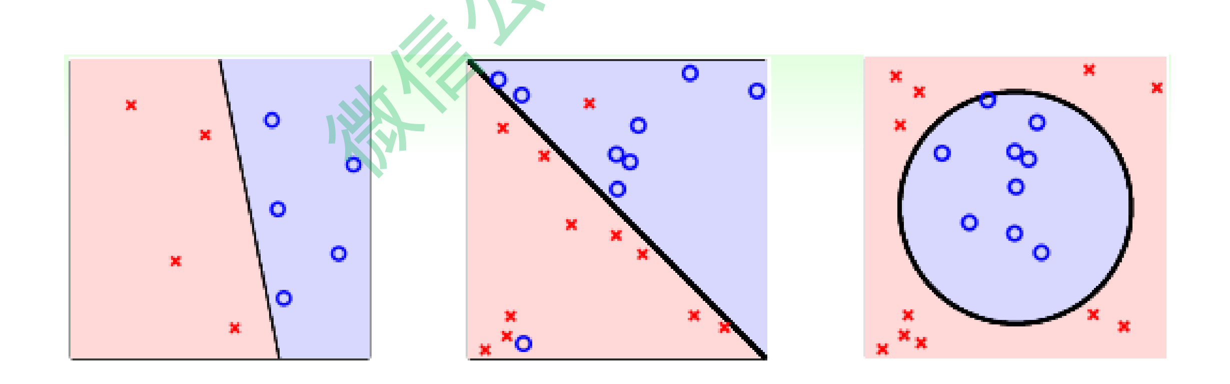
作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了解决线性分类问题的一个简单的方法：PLA。PLA能够在平面中选择一条直线将样本数据完全正确分类。而对于线性不可分的情况，可以使用Pocket Algorithm来处理。本节课将主要介绍一下机器学习有哪些种类，并进行归纳。

一、Learning with Different Output Space Y

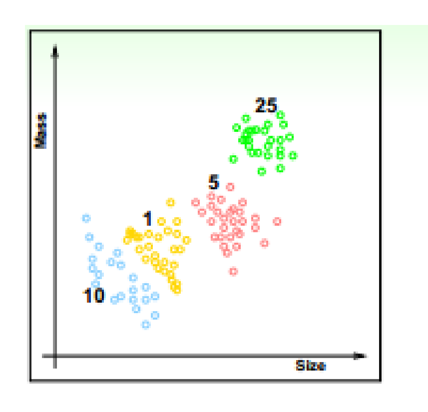
我们在上节课引入的银行根据用户个人情况判断是否给他发信用卡的例子，这是一个典型的二元分类（binary classification）问题。也就是说输出只有两个，一般y={-1, +1}，-1代表不发信用卡（负类），+1代表发信用卡（正类）。

二元分类的问题很常见，包括信用卡发放、垃圾邮件判别、患者疾病诊断、答案正确性估计等等。二元分类是机器学习领域非常核心和基本的问题。二元分类有线性模型也有非线性模型，根据实际问题情况，选择不同的模型。



除了二元分类，也有多元分类（Multiclass Classification）问题。顾名思义，多元分类的输出多于两个，y={1, 2, ... , K}, K>2. 一般多元分类的应用有数字识别、图片内容识别等等。



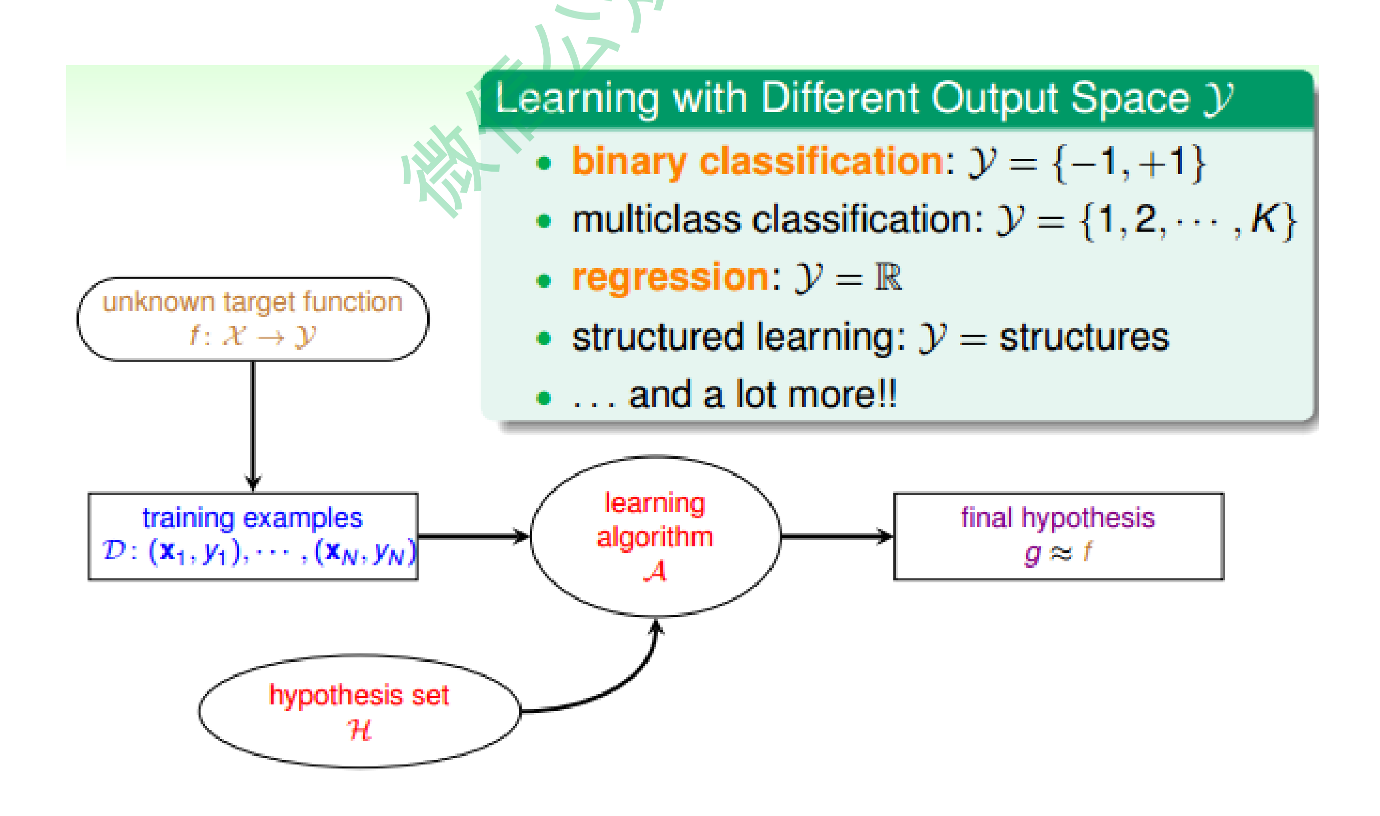


- classify US coins (1c, 5c, 10c, 25c) by (size, mass)
- $\mathcal{Y} = \{1c, 5c, 10c, 25c\}$, or $\mathcal{Y} = \{1, 2, \cdots, K\}$ (abstractly)
- binary classification: special case with $K = 2$

二元分类和多元分类都属于分类问题，它们的输出都是离散值。二对于另外一种情况，比如训练模型，预测房屋价格、股票收益多少等，这类问题的输出y=R，即范围在整个实数空间，是连续的。这类问题，我们把它叫做回归（Regression）。最简单的线性回归是一种典型的回归模型。

除了分类和回归问题，在自然语言处理等领域中，还会用到一种机器学习问题：结构化学习（Structured Learning）。结构化学习的输出空间包含了某种结构在里面，它的一些解法通常是从多分类问题延伸而来的，比较复杂。本系列课程不会详细介绍Structured Learning，有兴趣的读者可以自行对它进行更深入的研究。

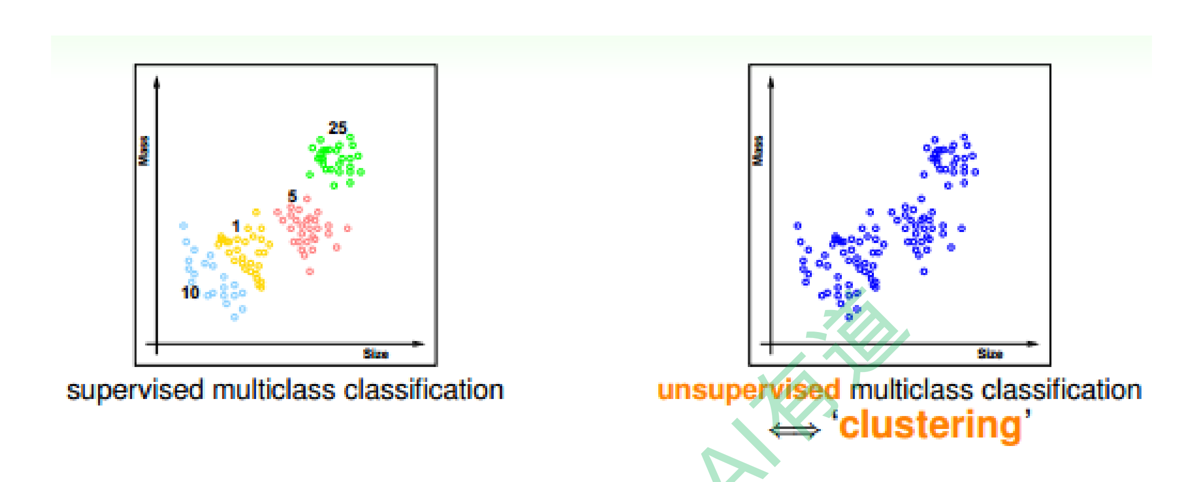
简单总结一下，机器学习按照输出空间划分的话，包括二元分类、多元分类、回归、结构化学习等不同的类型。其中二元分类和回归是最基础、最核心的两个类型，也是我们课程主要介绍的部分。



二、Learning with Different Data Label yn



如果我们拿到的训练样本D既有输入特征x，也有输出yn，那么我们把这种类型的学习称为监督式学习（Supervised Learning）。监督式学习可以是二元分类、多元分类或者是回归，最重要的是知道输出标签yn。与监督式学习相对立的另一种类型是非监督式学习（Unsupervised learning）。非监督式学习是没有输出标签yn的，典型的非监督式学习包括：聚类（clustering）问题，比如对网页上新闻的自动分类；密度估计，比如交通路况分析；异常检测，比如用户网络流量监测。通常情况下，非监督式学习更复杂一些，而且非监督的问题很多都可以使用监督式学习的一些算法思想来实现。

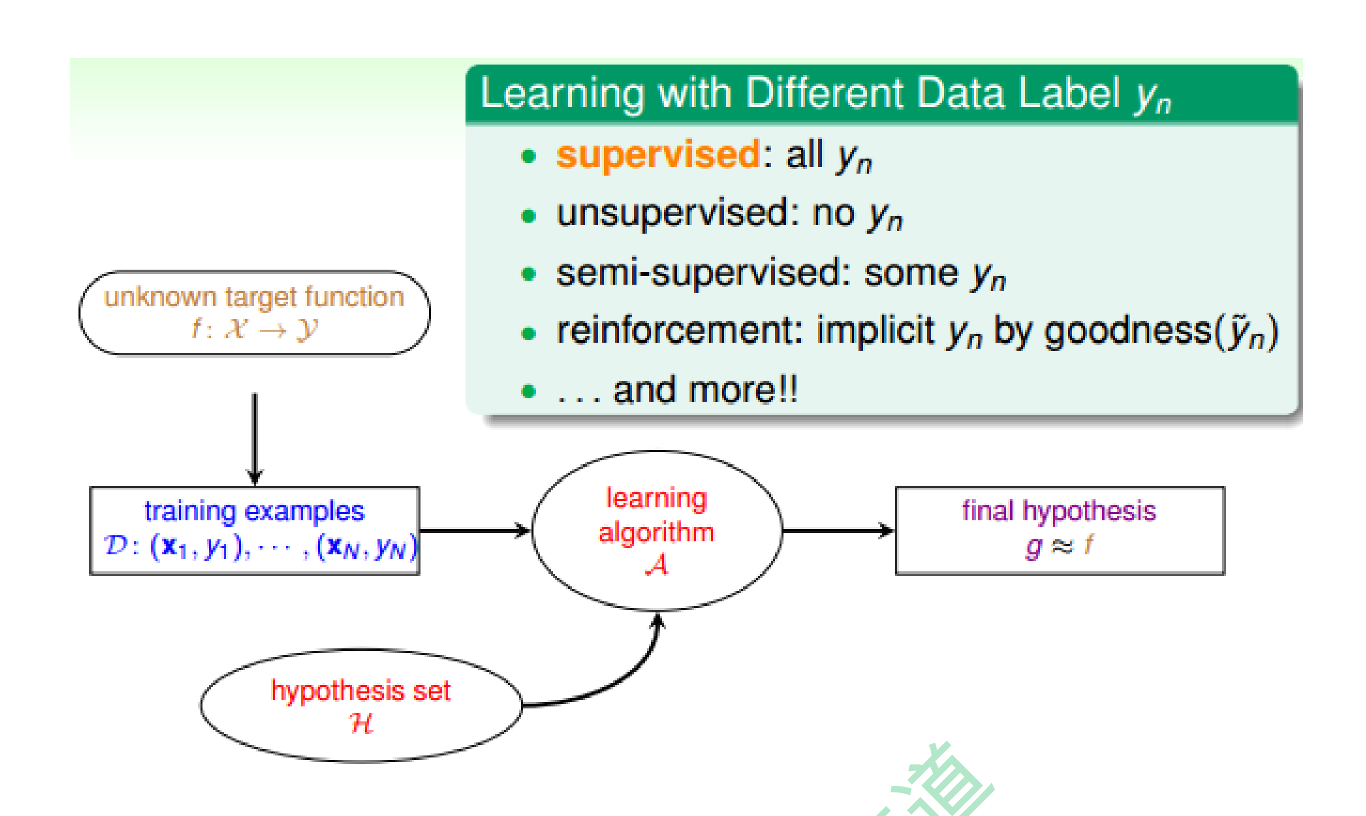


介于监督式和非监督式学习之间的叫做半监督式学习（Semi-supervised Learning）。顾名思义，半监督式学习就是说一部分数据有输出标签yn，而另一部分数据没有输出标签yn。在实际应用中，半监督式学习有时候是必须的，比如医药公司对某些药物进行检测，考虑到成本和实验人群限制等问题，只有一部分数据有输出标签yn。

监督式、非监督式、半监督式学习是机器学习领域三个主要类型。除此之外，还有一种非常重要的类型：增强学习（Reinforcement Learning）。增强学习中，我们给模型或系统一些输入，但是给不了我们希望的真实的输出y，根据模型的输出反馈，如果反馈结果良好，更接近真实输出，就给其正向激励，如果反馈结果不好，偏离真实输出，就给其反向激励。不断通过“反馈-修正”这种形式，一步一步让模型学习的更好，这就是增强学习的核心所在。增强学习可以类比成训练宠物的过程，比如我们要训练狗狗坐下，但是狗狗无法直接听懂我们的指令“sit down”。在训练过程中，我们给狗狗示意，如果它表现得好，我们就给他奖励，如果它做跟sit down完全无关的动作，我们就给它小小的惩罚。这样不断修正狗狗的动作，最终能让它按照我们的指令来行动。实际生活中，增强学习的例子也很多，比如根据用户点击、选择而不断改进的广告系统

简单总结一下，机器学习按照数据输出标签yn划分的话，包括监督式学习、非监督式学习、半监督式学习和增强学习等。其中，监督式学习应用最为广泛。





三、Learning with Different Protocol f(xn,yn)

按照不同的协议，机器学习可以分为三种类型：

- Batch Learning
- Online
- Active Learning

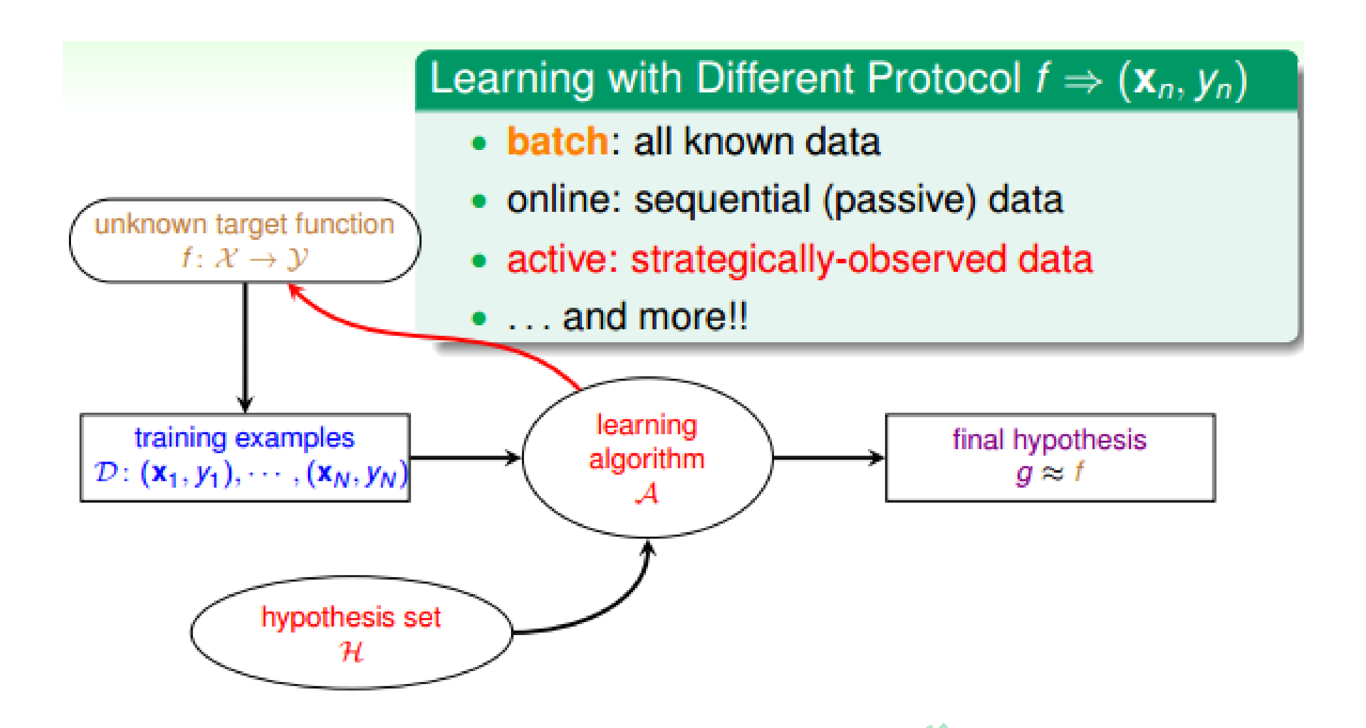
batch learning是一种常见的类型。batch learning获得的训练数据D是一批的，即一次性拿到整个D，对其进行学习建模，得到我们最终的机器学习模型。batch learning在实际应用中最为广泛。

online是一种在线学习模型，数据是实时更新的，根据数据一个个进来，同步更新我们的算法。比如在线邮件过滤系统，根据一封一封邮件的内容，根据当前算法判断是否为垃圾邮件，再根据用户反馈，及时更新当前算法。这是一个动态的过程。之前我们介绍的PLA和增强学习都可以使用online模型。

active learning是近些年来新出现的一种机器学习类型，即让机器具备主动问问题的能力，例如手写数字识别，机器自己生成一个数字或者对它不确定的手写字主动提问。active learning优势之一是在获取样本label比较困难的时候，可以节约时间和成本，只对一些重要的label提出需求。

简单总结一下，按照不同的协议，机器学习可以分为batch, online, active。这三种学习类型分别可以类比为：填鸭式，老师教学以及主动问问题。





四、Learning with Different Input Space X

上面几部分介绍的机器学习分类都是根据输出来分类的，比如根据输出空间进行分类，根据输出y的标记进行分类，根据取得数据和标记的方法进行分类。这部分，我们将谈谈输入X有哪些类型。

输入X的第一种类型就是concrete features。比如说硬币分类问题中硬币的尺寸、重量等；比如疾病诊断中的病人信息等具体特征。concrete features对机器学习来说最容易理解和使用。

第二种类型是raw features。比如说手写数字识别中每个数字所在图片的mxn维像素值；比如语音信号的频谱等。raw features一般比较抽象，经常需要人或者机器来转换为其对应的concrete features，这个转换的过程就是Feature Transform。

第三种类型是abstract features。比如某购物网站做购买预测时，提供给参赛者的是抽象加密过的资料编号或者ID，这些特征X完全是抽象的，没有实际的物理含义。所以对于机器学习来说是比较困难的，需要对特征进行更多的转换和提取。

简单总结一下，根据输入X类型不同，可以分为concetet, raw, abstract。将一些抽象的特征转换为具体的特征，是机器学习过程中非常重要的一个环节。在《机器学习技法》课程中，我们再详细介绍。



Learning with Different Input Space $\mathcal{X}$

- concrete: sophisticated (and related) physical meaning
- raw: simple physical meaning
- abstract: no (or little) physical meaning
- . . . and more!!

unknown target function $f: \mathcal{X} \to \mathcal{Y}$

training examples $\mathcal{D}: (\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$

learning algorithm $\mathcal{A}$

final hypothesis $g \approx f$

hypothesis set $\mathcal{H}$

五、总结：

本节课主要介绍了机器学习的类型，包括Out Space、Data Label、Protocol、Input Space四种类型。

- Learning with Different Output Space $\mathcal{Y}$
  [classification], [regression], structured
- Learning with Different Data Label y_n
  [supervised], un/semi-supervised, reinforcement
- Learning with Different Protocol $f \Rightarrow (\mathbf{x}_n, y_n)$
  [batch], online, active
- Learning with Different Input Space $\mathcal{X}$
  [concrete], raw, abstract

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程。



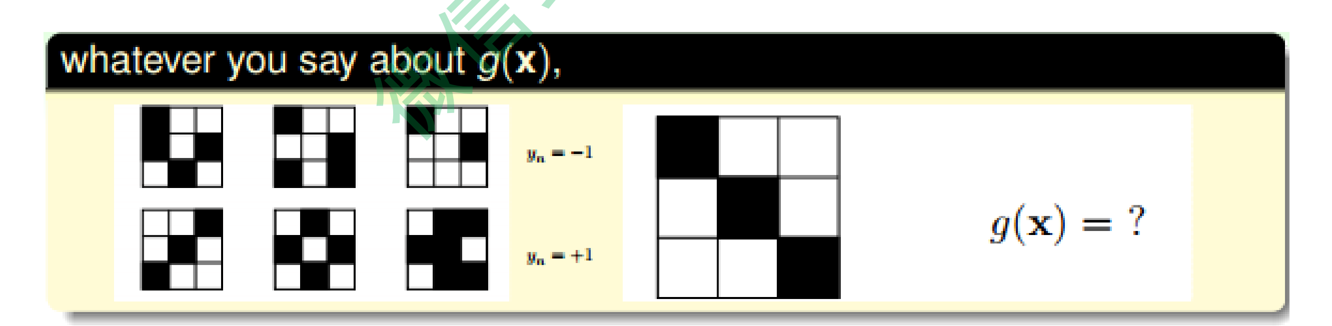
林轩田《机器学习基石》课程笔记4 -- Feasibility of Learning

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课，我们主要介绍了根据不同的设定，机器学习可以分为不同的类型。其中，监督式学习中的二元分类和回归分析是最常见的也是最重要的机器学习问题。本节课，我们将介绍机器学习的可行性，讨论问题是否可以使用机器学习来解决。

一、Learning is Impossible

首先，考虑这样一个例子，如下图所示，有3个label为-1的九宫格和3个label为+1的九宫格。根据这6个样本，提取相应label下的特征，预测右边九宫格是属于-1还是+1？结果是，如果依据对称性，我们会把它归为+1；如果依据九宫格左上角是否是黑色，我们会把它归为-1。除此之外，还有根据其它不同特征进行分类，得到不同结果的情况。而且，这些分类结果貌似都是正确合理的，因为对于6个训练样本来说，我们选择的模型都有很好的分类效果。



truth $f(\mathbf{x}) = +1$ because ...	truth $f(\mathbf{x}) = -1$ because ...
• symmetry ⇔ +1	• left-top black ⇔ -1
• (black or white count = 3) or (black count = 4 and middle-top black) ⇔ +1	• middle column contains at most 1 black and right-top white ⇔ -1

再来看一个比较数学化的二分类例子，输入特征x是二进制的、三维的，对应有8种输入，其中训练样本D有5个。那么，根据训练样本对应的输出y，假设有8个hypothesis，这8个hypothesis在D上，对5个训练样本的分类效果效果都完全正确。



是在另外3个测试数据上，不同的hypothesis表现有好有坏。在已知数据D上，$g \approx f$；但是在D以外的未知数据上，$g \approx f$不一定成立。而机器学习目的，恰恰是希望我们选择的模型能在未知数据上的预测与真实结果是一致的，而不是在已知的数据集D上寻求最佳效果。

	x	y	g	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
$\mathcal{D}$	000	○	○	○	○	○	○	○	○	○	○
	001	×	×	×	×	×	×	×	×	×	×
	010	×	×	×	×	×	×	×	×	×	×
	011	○	○	○	○	○	○	○	○	○	○
	100	×	×	×	×	×	×	×	×	×	×
	101		?	○	○	○	○	×	×	×	×
	110		?	○	○	×	×	○	○	×	×
	111		?	○	×	○	×	○	×	○	×

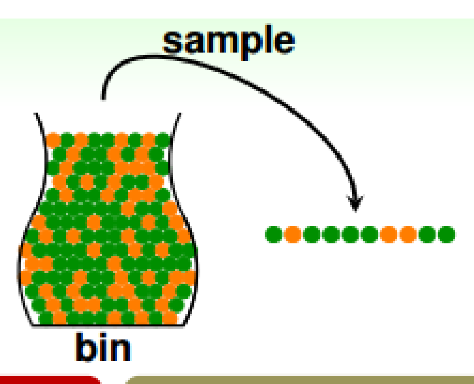
这个例子告诉我们，我们想要在D以外的数据中更接近目标函数似乎是做不到的，只能保证对D有很好的分类结果。机器学习的这种特性被称为没有免费午餐（No Free Lunch）定理。NFL定理表明没有一个学习算法可以在任何领域总是产生最准确的学习器。不管采用何种学习算法，至少存在一个目标函数，能够使得随机猜测算法是更好的算法。平常所说的一个学习算法比另一个算法更“优越”，效果更好，只是针对特定的问题，特定的先验信息，数据的分布，训练样本的数目，代价或奖励函数等。从这个例子来看，NFL说明了无法保证一个机器学习算法在D以外的数据集上一定能分类或预测正确，除非加上一些假设条件，我们以后会介绍。

二、Probability to the Rescue

从上一节得出的结论是：在训练集D以外的样本上，机器学习的模型是很难，似乎做不到正确预测或分类的。那是否有一些工具或者方法能够对未知的目标函数f做一些推论，让我们的机器学习模型能够变得有用呢？

如果有一个装有很多（数量很大数不过来）橙色球和绿色球的罐子，我们能不能推断橙色球的比例u？统计学上的做法是，从罐子中随机取出N个球，作为样本，计算这N个球中橙色球的比例v，那么就估计出罐子中橙色球的比例约为v。





bin

assume

orange probability = μ,
green probability = $1 - \mu$,

with μ **unknown**

sample

N marbles sampled independently, with

orange fraction = ν,
green fraction = $1 - \nu$,

now ν **known**

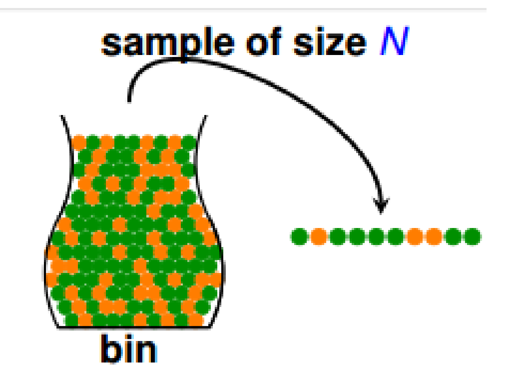
这种随机抽取的做法能否说明罐子里橙色球的比例一定是v呢？答案是否定的。但是从概率的角度来说，样本中的v很有可能接近我们未知的u。下面从数学推导的角度来看v与u是否相近。

已知u是罐子里橙色球的比例，v是N个抽取的样本中橙色球的比例。当N足够大的时候，v接近于u。这就是Hoeffding's inequality：

$$P[|v-u|>\epsilon] \leq 2exp(-2\epsilon^2 N)$$

Hoeffding不等式说明当N很大的时候，v与u相差不会很大，它们之间的差值被限定在ϵ之内。我们把结论v=u称为probably approximately correct(PAC)。

- valid for all N and ϵ
- does not depend on μ,
  no need to 'know' μ
- larger sample size N or
  looser gap ϵ
  $\Longrightarrow$ higher probability for '$\nu \approx \mu$'



if **large** N, can **probably** infer
unknown μ by known ν

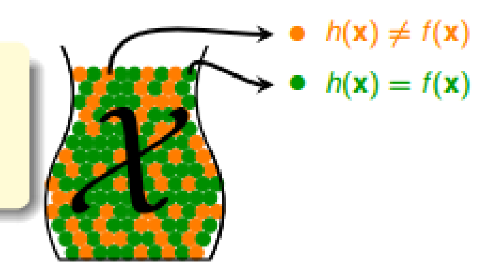
三、Connection to Learning



下面，我们将罐子的内容对应到机器学习的概念上来。机器学习中hypothesis与目标函数相等的可能性，类比于罐子中橙色球的概率问题；罐子里的一颗颗弹珠类比于机器学习样本空间的x；橙色的弹珠类比于h(x)与f不相等；绿色的弹珠类比于h(x)与f相等；从罐子中抽取的N个球类比于机器学习的训练样本D，且这两种抽样的样本与总体样本之间都是独立同分布的。所以呢，如果样本N够大，且是独立同分布的，那么，从样本中$h(x) \neq f(x)$的概率就能推导在抽样样本外的所有样本中$h(x) \neq f(x)$的概率是多少。

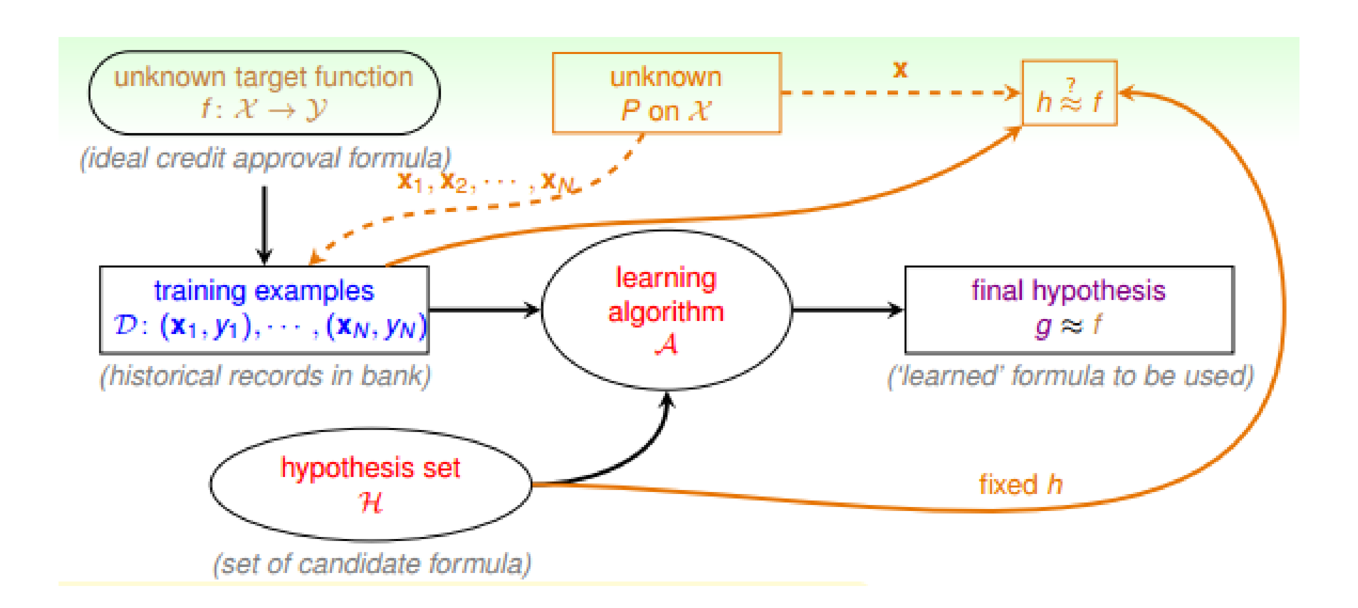
bin	learning
unknown orange prob. μ	fixed hypothesis $h(\mathbf{x}) \stackrel{?}{=}$ target $f(\mathbf{x})$
marble $\bullet \in$ bin	$\mathbf{x} \in \mathcal{X}$
orange $\bullet$	h is wrong $\Leftrightarrow h(\mathbf{x}) \neq f(\mathbf{x})$
green $\bullet$	h is right $\Leftrightarrow h(\mathbf{x}) = f(\mathbf{x})$
size-N sample from bin of i.i.d. marbles	check h on $\mathcal{D} = \{(\mathbf{x}_n, \underbrace{y_n}_{f(\mathbf{x}_n)})\}$ with i.i.d. $\mathbf{x}_n$

if **large** N & **i.i.d.** $\mathbf{x}_n$, can **probably** infer
unknown $[\![h(\mathbf{x}) \neq f(\mathbf{x})]\!]$ probability
by known $[\![h(\mathbf{x}_n) \neq y_n]\!]$ fraction



映射中最关键的点是讲抽样中橙球的概率理解为样本数据集D上h(x)错误的概率，以此推算出在所有数据上h(x)错误的概率，这也是机器学习能够工作的本质，即我们为啥在采样数据上得到了一个假设，就可以推到全局呢？因为两者的错误率是PAC的，只要我们保证前者小，后者也就小了。





这里我们引入两个值$E_{in}(h)$和$E_{out}(h)$。$E_{in}(h)$表示在抽样样本中，h(x)与y_n不相等的概率；$E_{out}(h)$表示实际所有样本中，h(x)与f(x)不相等的概率是多少。

$$\text{unknown } E_{\text{out}}(\mathbf{h}) = \underset{\mathbf{x}\sim P}{\mathcal{E}} [\![h(\mathbf{x}) \neq f(\mathbf{x})]\!]$$

$$\text{by known } E_{\text{in}}(\mathbf{h}) = \frac{1}{N} \sum_{n=1}^{N} [\![h(\mathbf{x}_n) \neq y_n]\!].$$

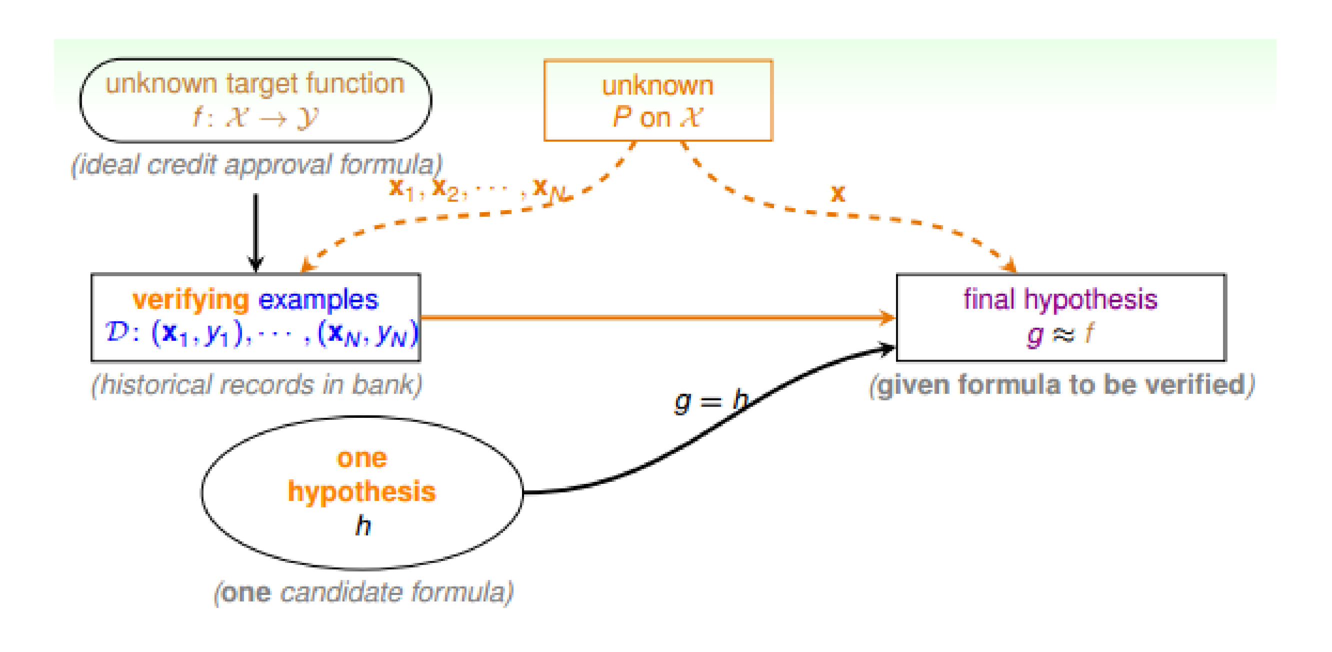
同样，它的Hoeffding's inequality可以表示为：

$$P[|E_{in}(h) - E_{out}(h)| > \epsilon] \leq 2exp(-2\epsilon^2 N)$$

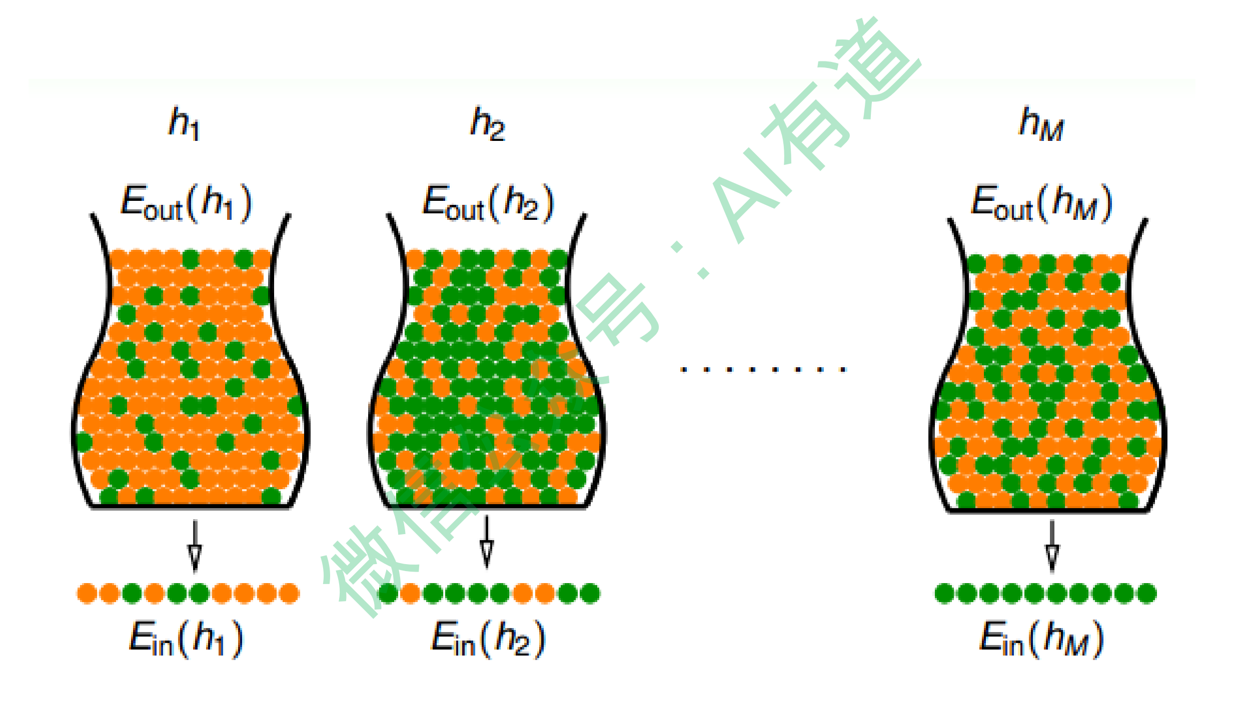
该不等式表明，$E_{in}(h) = E_{out}(h)$也是PAC的。如果$E_{in}(h) \approx E_{out}(h)$，$E_{in}(h)$很小，那么就能推断出$E_{out}(h)$很小，也就是说在该数据分布P下，h与f非常接近，机器学习的模型比较准确。

一般地，h如果是固定的，N很大的时候，$E_{in}(h) \approx E_{out}(h)$，但是并不意味着$g \approx f$。因为h是固定的，不能保证$E_{in}(h)$足够小，即使$E_{in}(h) \approx E_{out}(h)$，也可能使$E_{out}(h)$偏大。所以，一般会通过演算法A，选择最好的h，使$E_{in}(h)$足够小，从而保证$E_{out}(h)$很小。固定的h，使用新数据进行测试，验证其错误率是多少。





四、Connection to Real Learning



假设现在有很多罐子M个（即有M个hypothesis），如果其中某个罐子抽样的球全是绿色，那是不是应该选择这个罐子呢？我们先来看这样一个例子：150个人抛硬币，那么其中至少有一个人连续5次硬币都是正面朝上的概率是

$$1 - (\frac{31}{32})^{150} > 99\%$$

可见这个概率是很大的，但是能否说明5次正面朝上的这个硬币具有代表性呢？答案是否定的！并不能说明该硬币单次正面朝上的概率很大，其实都是0.5。一样的道理，抽到全是绿色求的时候也不能一定说明那个罐子就全是绿色球。当罐子数目很多或者抛硬币的人数很多的时候，可能引发Bad Sample，Bad Sample就是E_{in}和E_{out}差别很大，即选择过多带来的负面影响，选择过多会恶化不好的情形。



根据许多次抽样的到的不同的数据集D，Hoeffding's inequality保证了大多数的D都是比较好的情形（即对于某个h，保证$E_{in} \approx E_{out}$），但是也有可能出现Bad Data，即E_{in}和E_{out}差别很大的数据集D，这是小概率事件。

	$\mathcal{D}_1$	$\mathcal{D}_2$	...	$\mathcal{D}_{1126}$	...	$\mathcal{D}_{5678}$	Hoeffding
h_1	BAD					BAD	$\mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_1] \leq \ldots$
h_2		BAD					$\mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_2] \leq \ldots$
h_3	BAD	BAD				BAD	$\mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_3] \leq \ldots$
...							
h_M	BAD					BAD	$\mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_M] \leq \ldots$
all	BAD	BAD				BAD	?

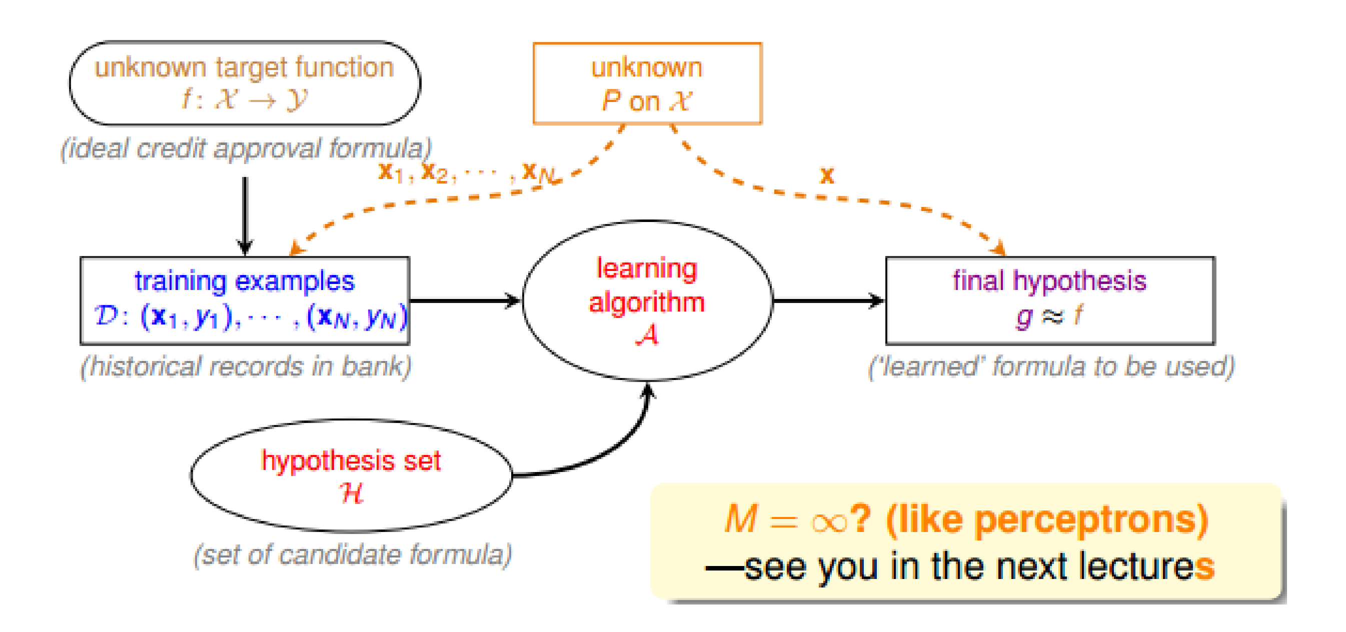
也就是说，不同的数据集D_n，对于不同的hypothesis，有可能成为Bad Data。只要D_n在某个hypothesis上是Bad Data，那么D_n就是Bad Data。只有当D_n在所有的hypothesis上都是好的数据，才说明D_n不是Bad Data，可以自由选择演算法A进行建模。那么，根据Hoeffding's inequality，Bad Data的上界可以表示为连级（union bound）的形式：

$$
\begin{aligned}
& \mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D}] \\
= \; & \mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_1 \textbf{ or } \text{BAD } \mathcal{D} \text{ for } h_2 \textbf{ or } \ldots \textbf{ or } \text{BAD } \mathcal{D} \text{ for } h_M] \\
\leq \; & \mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_1] + \mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_2] + \ldots + \mathbb{P}_{\mathcal{D}}[\text{BAD } \mathcal{D} \text{ for } h_M] \\
& \text{(union bound)} \\
\leq \; & 2\exp\left(-2\epsilon^2 N\right) + 2\exp\left(-2\epsilon^2 N\right) + \ldots + 2\exp\left(-2\epsilon^2 N\right) \\
= \; & 2M\exp\left(-2\epsilon^2 N\right)
\end{aligned}
$$

其中，M是hypothesis的个数，N是样本D的数量，ϵ是参数。该union bound表明，当M有限，且N足够大的时候，Bad Data出现的概率就更低了，即能保证D对于所有的h都有$E_{in} \approx E_{out}$，满足PAC，演算法A的选择不受限制。那么满足这种union bound的情况，我们就可以和之前一样，选取一个合理的演算法（PLA/pocket），选择使E_{in}最小的h_m作为矩g，一般能够保证$g \approx f$，即有不错的泛化能力。

所以，如果hypothesis的个数M是有限的，N足够大，那么通过演算法A任意选择一个矩g，都有$E_{in} \approx E_{out}$成立；同时，如果找到一个矩g，使$E_{in} \approx 0$，PAC就能保证$E_{out} \approx 0$。至此，就证明了机器学习是可行的。





但是，如上面的学习流程图右下角所示，如果M是无数个，例如之前介绍的PLA直线有无数条，是否这些推论就不成立了呢？是否机器就不能进行学习呢？这些内容和问题，我们下节课再介绍。

五、总结

本节课主要介绍了机器学习的可行性。首先引入NFL定理，说明机器学习无法找到一个矩g能够完全和目标函数f一样。接着介绍了可以采用一些统计上的假设，例如Hoeffding不等式，建立E_{in}和E_{out}的联系，证明对于某个h，当N足够大的时候，E_{in}和E_{out}是PAC的。最后，对于h个数很多的情况，只要有h个数M是有限的，且N足够大，就能保证$E_{in} \approx E_{out}$，证明机器学习是可行的。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程。



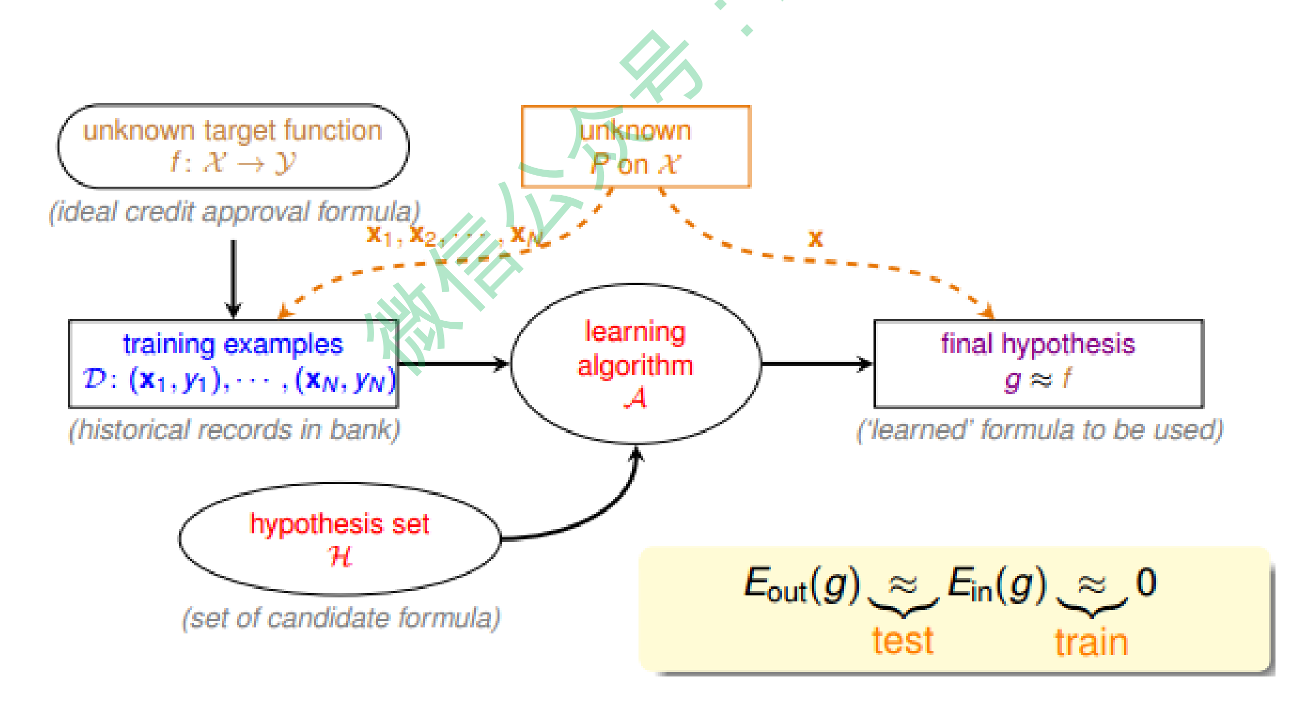
林轩田《机器学习基石》课程笔记5 -- Training versus Testing

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课，我们主要介绍了机器学习的可行性。首先，由NFL定理可知，机器学习貌似是不可行的。但是，随后引入了统计学知识，如果样本数据足够大，且hypothesis个数有限，那么机器学习一般就是可行的。本节课将讨论机器学习的核心问题，严格证明为什么机器可以学习。从上节课最后的问题出发，即当hypothesis的个数是无限多的时候，机器学习的可行性是否仍然成立？

一、Recap and Preview

我们先来看一下基于统计学的机器学习流程图：



该流程图中，训练样本D和最终测试h的样本都是来自同一个数据分布，这是机器能够学习的前提。另外，训练样本D应该足够大，且hypothesis set的个数是有限的，这样根据霍夫丁不等式，才不会出现Bad Data，保证$E_{in} \approx E_{out}$，即有很好的泛化能力。同时，通过训练，得到使E_{in}最小的h，作为模型最终的矩g，g接近于目标函数。

这里，我们总结一下前四节课的主要内容：第一节课，我们介绍了机器学习的定义，目标是找出最好的矩g，使$g \approx f$，保证$E_{out}(g) \approx 0$；第二节课，我们介绍了如何



$E_{in} \approx 0$，可以使用PLA、pocket等演算法来实现；第三节课，我们介绍了机器学习的分类，我们的训练样本是批量数据（batch），处理监督式（supervised）二元分类（binary classification）问题；第四节课，我们介绍了机器学习的可行性，通过统计学知识，把$E_{in}(g)$与$E_{out}(g)$联系起来，证明了在一些条件假设下，$E_{in}(g) \approx E_{out}(g)$成立。

for batch & supervised binary classification (lecture 3), $g \approx f$ (lecture 1) $\iff E_{out}(g) \approx 0$

achieved through $E_{out}(g) \approx E_{in}(g)$ (lecture 4) and $E_{in}(g) \approx 0$ (lecture 2)

这四节课总结下来，我们把机器学习的主要目标分成两个核心的问题：

- $E_{in}(g) \approx E_{out}(g)$
- $E_{in}(g)$足够小

上节课介绍的机器学习可行的一个条件是hypothesis set的个数M是有限的，那M跟上面这两个核心问题有什么联系呢?

我们先来看一下，当M很小的时候，由上节课介绍的霍夫丁不等式，得到$E_{in}(g) \approx E_{out}(g)$，即能保证第一个核心问题成立。但M很小时，演算法A可以选择的hypothesis有限，不一定能找到使$E_{in}(g)$足够小的hypothesis，即不能保证第二个核心问题成立。当M很大的时候，同样由霍夫丁不等式，$E_{in}(g)$与$E_{out}(g)$的差距可能比较大，第一个核心问题可能不成立。而M很大，使的演算法A的可以选择的hypothesis就很多，很有可能找到一个hypothesis，使$E_{in}(g)$足够小，第二个核心问题可能成立。

1. can we make sure that $E_{out}(g)$ is close enough to $E_{in}(g)$?
2. can we make $E_{in}(g)$ small enough?

small M	large M
1. Yes!, $\mathbb{P}[\textbf{BAD}] \le 2 \cdot M \cdot \exp(\ldots)$	1. No!, $\mathbb{P}[\textbf{BAD}] \le 2 \cdot M \cdot \exp(\ldots)$
2. No!, too few choices	2. Yes!, many choices



从上面的分析来看，M的选择直接影响机器学习两个核心问题是否满足，M不能太大也不能太小。那么如果M无限大的时候，是否机器就不可以学习了呢？例如PLA算法中直线是无数条的，但是PLA能够很好地进行机器学习，这又是为什么呢？如果我们能将无限大的M限定在一个有限的m_H内，问题似乎就解决了。

二、Effective Number of Line

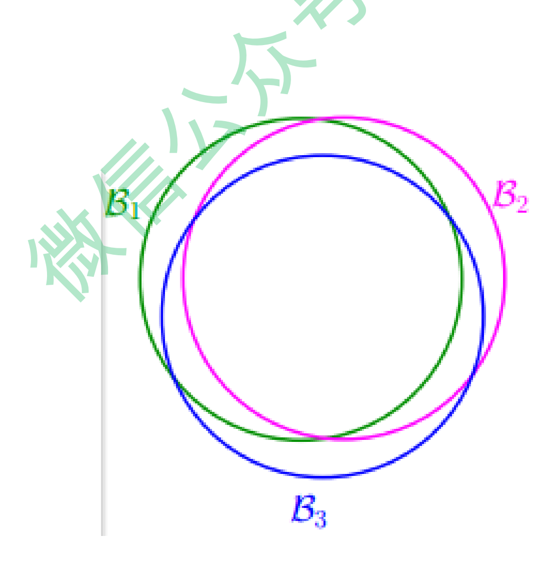
我们先看一下上节课推导的霍夫丁不等式：

$$P[|E_{in}(g)-E_{out}(g)|>\epsilon]\leq 2\cdot M\cdot exp(-2\epsilon^2N)$$

其中，M表示hypothesis的个数。每个hypothesis下的BAD events B_m级联的形式满足下列不等式：

$$P[B_1\ or\ B_2\ or\ \cdots B_M]\leq P[B_1]+P[B_2]+\cdots+P[B_M]$$

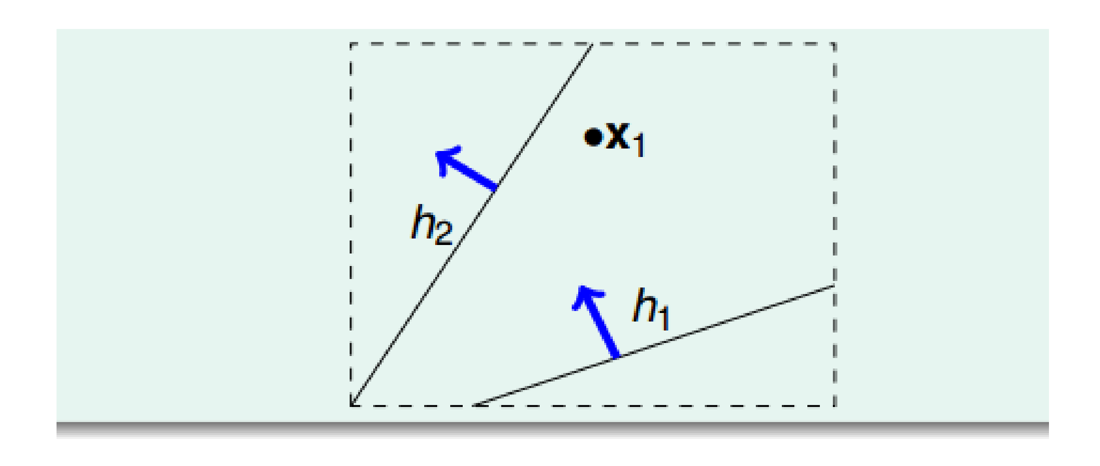
当$M=\infty$时，上面不等式右边值将会很大，似乎说明BAD events很大，$E_{in}(g)$与$E_{out}(g)$也并不接近。但是BAD events B_m级联的形式实际上是扩大了上界，union bound过大。这种做法假设各个hypothesis之间没有交集，这是最坏的情况，可是实际上往往不是如此，很多情况下，都是有交集的，也就是说M实际上没那么大，如下图所示：



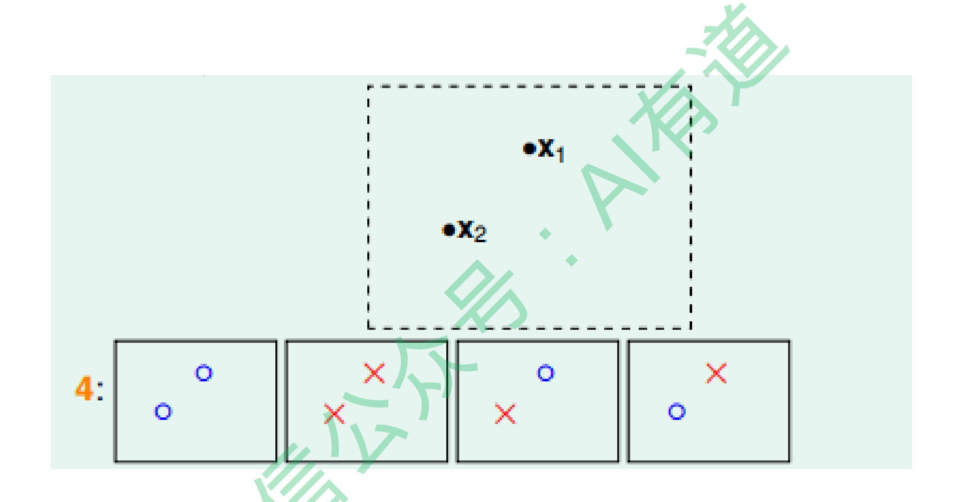
也就是说union bound被估计过高了（over-estimating）。所以，我们的目的是找出不同BAD events之间的重叠部分，也就是将无数个hypothesis分成有限个类别。

如何将无数个hypothesis分成有限类呢？我们先来看这样一个例子，假如平面上用直线将点分开，也就跟PLA一样。如果平面上只有一个点x1，那么直线的种类有两种：一种将x1划为+1，一种将x1划为-1：

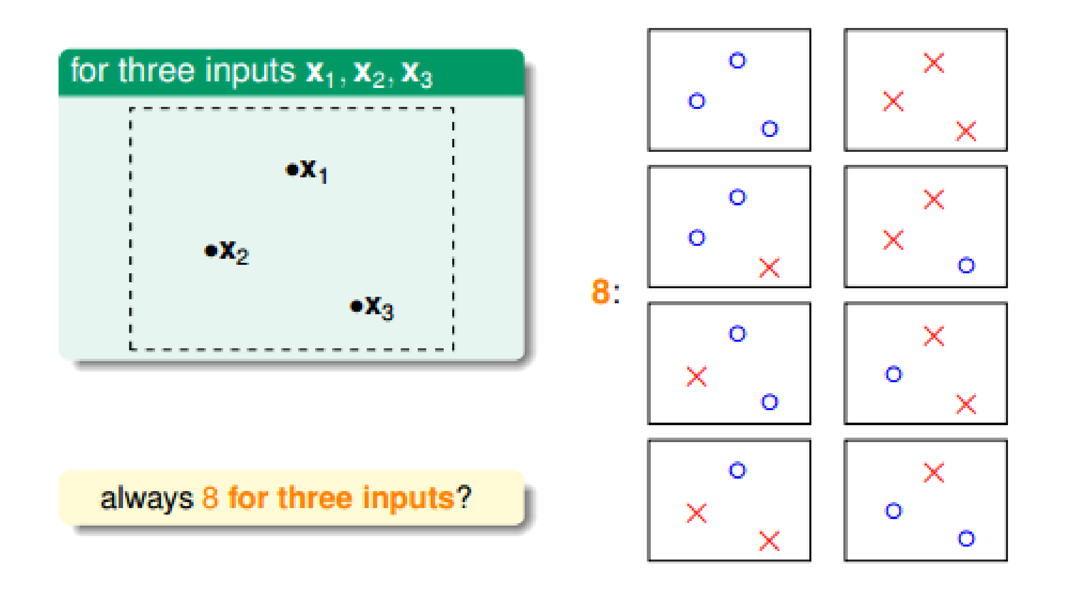




如果平面上有两个点x1、x2，那么直线的种类共4种：x1、x2都为+1，x1、x2都为-1，x1为+1且x2为-1，x1为-1且x2为+1：

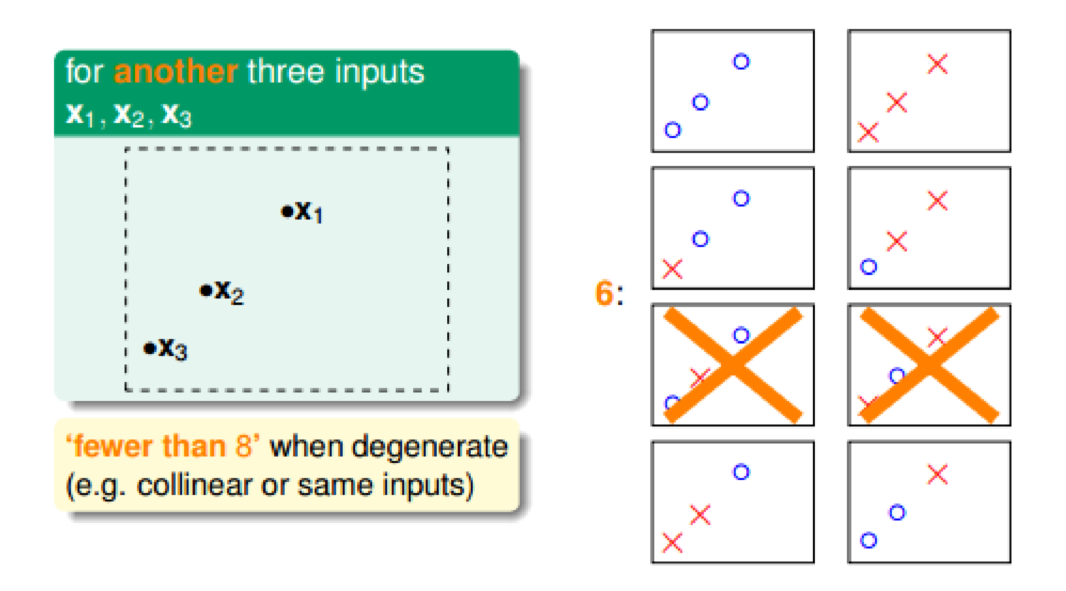


如果平面上有三个点x1、x2、x3，那么直线的种类共8种：

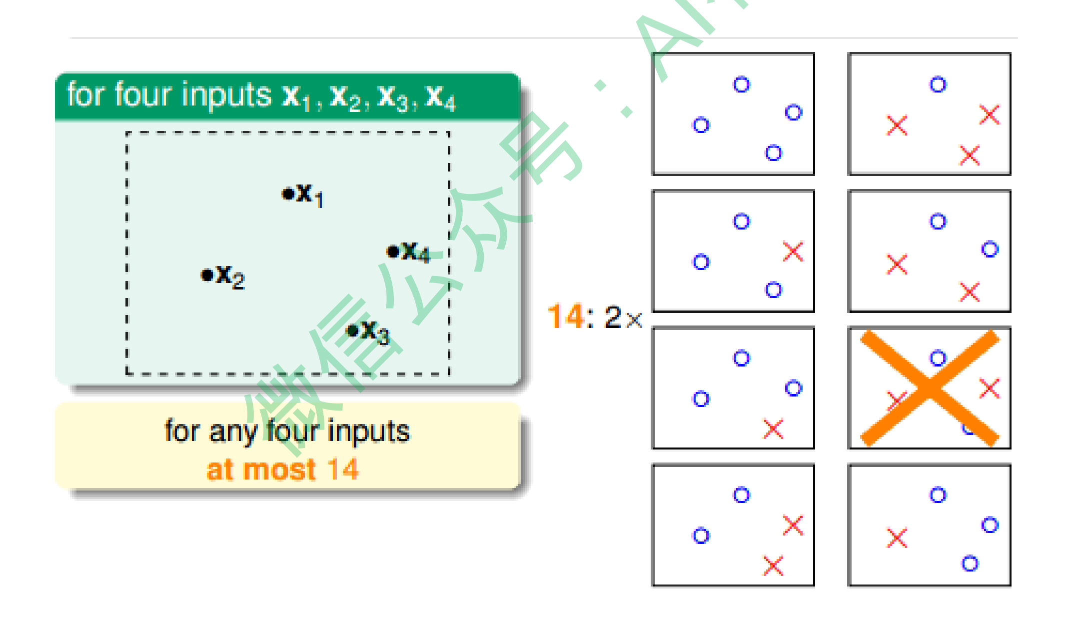


但是，在三个点的情况下，也会出现不能用一条直线划分的情况：





也就是说，对于平面上三个点，不能保证所有的8个类别都能被一条直线划分。那如果是四个点x1、x2、x3、x4，我们发现，平面上找不到一条直线能将四个点组成的16个类别完全分开，最多只能分开其中的14类，即直线最多只有14种：



经过分析，我们得到平面上线的种类是有限的，1个点最多有2种线，2个点最多有4种线，3个点最多有8种线，4个点最多有14（$< 2^4$）种线等等。我们发现，有效直线的数量总是满足$\leq 2^N$，其中，N是点的个数。所以，如果我们可以用effective(N)代替M，霍夫丁不等式可以写成：

$$P[|E_{in}(g) - E_{out}(g)| > \epsilon] \leq 2 \cdot effective(N) \cdot exp(-2\epsilon^2 N)$$

已知effective(N)<2^N，如果能够保证effective(N)<<2^N，即不等式右边接近于零，那么即使M无限大，直线的种类也很有限，机器学习也是可能的。



- must be $\leq 2^N$ (why?)
- finite 'grouping' of infinitely-many lines $\in \mathcal{H}$
- wish:

$$\mathbb{P}\left[\left|E_{\text{in}}(g) - E_{\text{out}}(g)\right| > \epsilon\right] \leq 2 \cdot \text{effective}(N) \cdot \exp\left(-2\epsilon^2 N\right)$$

lines in 2D

N	effective(N)
1	2
2	4
3	8
4	$14 < 2^N$

if ① effective(N) can replace M and
② effective(N) $\ll 2^N$
learning possible with infinite lines :-)

三、Effective Number of Hypotheses

接下来先介绍一个新名词：二分类（dichotomy）。dichotomy就是将空间中的点（例如二维平面）用一条直线分成正类（蓝色o）和负类（红色x）。令H是将平面上的点用直线分开的所有hypothesis h的集合，dichotomy H与hypotheses H的关系是：hypotheses H是平面上所有直线的集合，个数可能是无限个，而dichotomy H是平面上能将点完全用直线分开的直线种类，它的上界是2^N。接下来，我们要做的就是尝试用dichotomy代替M。

- call

$$h(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N) = (h(\mathbf{x}_1), h(\mathbf{x}_2), \ldots, h(\mathbf{x}_N)) \in \{\times, \circ\}^N$$

  a **dichotomy**: hypothesis 'limited' to the eyes of $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$
- $\mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$:
  all dichotomies 'implemented' by $\mathcal{H}$ on $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$

	hypotheses $\mathcal{H}$	dichotomies $\mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)$
e.g.	all lines in $\mathbb{R}^2$	$\{\circ\circ\circ\circ, \circ\circ\circ\times, \circ\circ\times\times, \ldots\}$
size	possibly infinite	upper bounded by 2^N

再介绍一个新的名词：成长函数（growth function），记为$m_H(H)$。成长函数的定义是：对于由N个点组成的不同集合中，某集合对应的dichotomy最大，那么这个dichotomy值就是$m_H(H)$，它的上界是2^N：



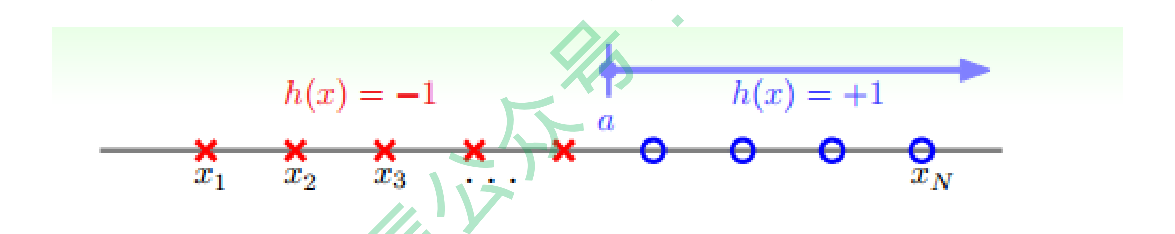
$$m_{\mathcal{H}}(N) = \max_{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N \in \mathcal{X}} |\mathcal{H}(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N)|$$

成长函数其实就是我们之前讲的effective lines的数量最大值。根据成长函数的定义，二维平面上，$m_H(H)$随N的变化关系是：

lines in 2D

N	$m_{\mathcal{H}}(N)$
1	2
2	4
3	$\max(\ldots, 6, 8) = 8$
4	$14 < 2^N$

接下来，我们讨论如何计算成长函数。先看一个简单情况，一维的Positive Rays：



若有N个点，则整个区域可分为N+1段，很容易得到其成长函数$m_H(N) = N + 1$。注意当N很大时，$(N + 1) << 2^N$，这是我们希望看到的。

另一种情况是一维的Positive Intervals：

$h(x) = -1$ $h(x) = +1$ $h(x) = -1$

x_1 x_2 x_3 ... x_N

它的成长函数可以由下面推导得出：



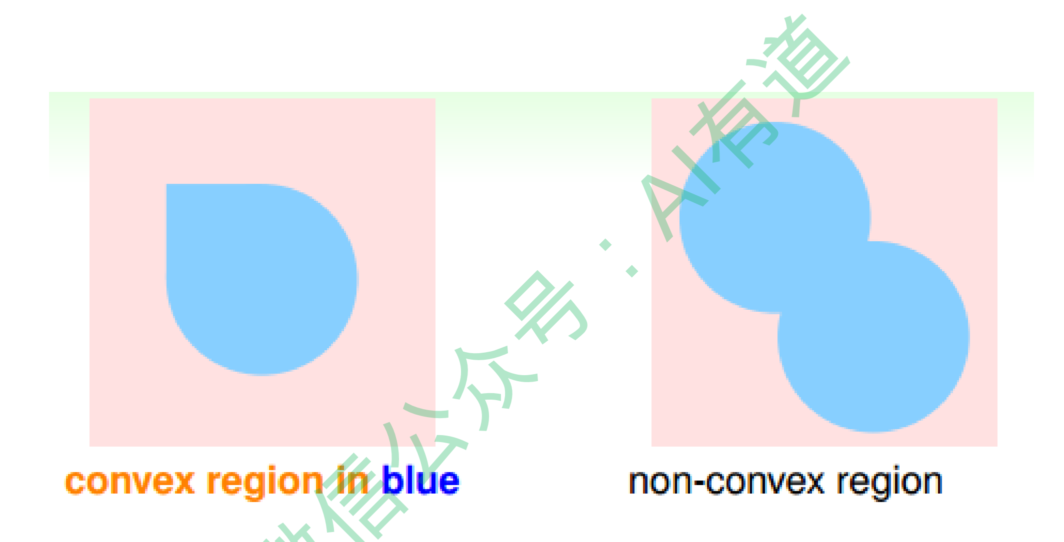
one dichotomy for each 'interval kind'

$$m_{\mathcal{H}}(N) = \underbrace{\binom{N+1}{2}}_{\text{interval ends in } N+1 \text{ spots}} + \underbrace{1}_{\text{all } \times}$$

$$= \frac{1}{2}N^2 + \frac{1}{2}N + 1$$

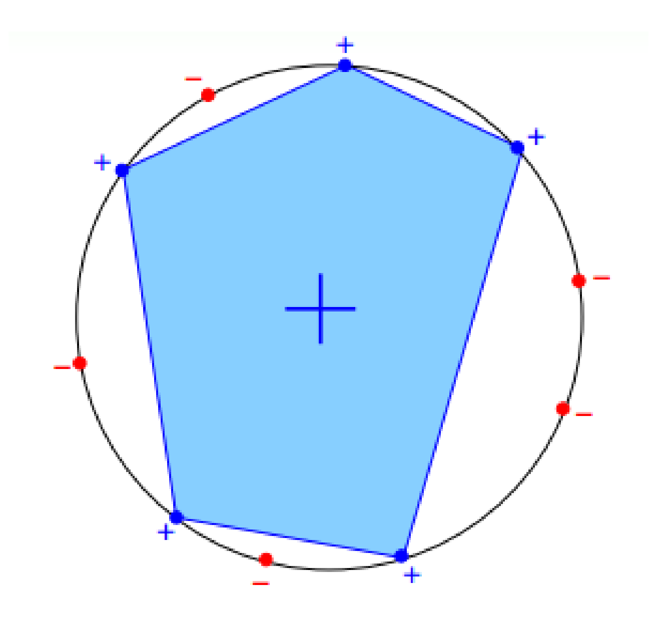
这种情况下，$m_H(N) = \frac{1}{2}N^2 + \frac{1}{2}N + 1 << 2^N$，在N很大的时候，仍然是满足的。

再来看这个例子，假设在二维空间里，如果hypothesis是凸多边形或类圆构成的封闭曲线，如下图所示，左边是convex的，右边不是convex的。那么，它的成长函数是多少呢？



当数据集D按照如下的凸分布时，我们很容易计算得到它的成长函数$m_H = 2^N$。这种情况下，N个点所有可能的分类情况都能够被hypotheses set覆盖，我们把这种情形称为shattered。也就是说，如果能够找到一个数据分布集，hypotheses set对N个输入所有的分类情况都做得到，那么它的成长函数就是2^N。





四、Break Point

上一小节，我们介绍了四种不同的成长函数，分别是：

- positive rays: $m_{\mathcal{H}}(N) = N + 1$
- positive intervals: $m_{\mathcal{H}}(N) = \frac{1}{2}N^2 + \frac{1}{2}N + 1$
- convex sets: $m_{\mathcal{H}}(N) = 2^N$
- 2D perceptrons: $m_{\mathcal{H}}(N) < 2^N$ **in some cases**

其中，positive rays和positive intervals的成长函数都是polynomial的，如果用m_H代替M的话，这两种情况是比较好的。而convex sets的成长函数是exponential的，即等于M，并不能保证机器学习的可行性。那么，对于2D perceptrons，它的成长函数究竟是polynomial的还是exponential的呢？

对于2D perceptrons，我们之前分析了3个点，可以做出8种所有的dichotomy，而4个点，就无法做出所有16个点的dichotomy了。所以，我们就把4称为2D perceptrons的break point（5、6、7等都是break point）。令有k个点，如果k大于等于break point时，它的成长函数一定小于2的k次方。

根据break point的定义，我们知道满足$m_H(k) \neq 2^k$的k的最小值就是break point。对于我们之前介绍的四种成长函数，他们的break point分别是：



- positive rays: $m_{\mathcal{H}}(N) = N + 1 = O(N)$
  break point at 2
- positive intervals: $m_{\mathcal{H}}(N) = \frac{1}{2}N^2 + \frac{1}{2}N + 1 = O(N^2)$
  break point at 3
- convex sets: $m_{\mathcal{H}}(N) = 2^N$
  no break point
- 2D perceptrons: $m_{\mathcal{H}}(N) < 2^N$ in some cases
  break point at 4

通过观察，我们猜测成长函数可能与break point存在某种关系：对于convex sets，没有break point，它的成长函数是2的N次方；对于positive rays，break point k=2，它的成长函数是O(N)；对于positive intervals，break point k=3，它的成长函数是$O(N^2)$。则根据这种推论，我们猜测2D perceptrons，它的成长函数$m_H(N) = O(N^{k-1})$ 。如果成立，那么就可以用m_H代替M，就满足了机器能够学习的条件。关于上述猜测的证明，我们下节课再详细介绍。

五、总结

本节课，我们更深入地探讨了机器学习的可行性。我们把机器学习拆分为两个核心问题：$E_{in}(g) \approx E_{out}(g)$和$E_{in}(g) \approx 0$。对于第一个问题，我们探讨了M个hypothesis到底可以划分为多少种，也就是成长函数m_H。并引入了break point的概念，给出了break point的计算方法。下节课，我们将详细论证对于2D perceptrons，它的成长函数与break point是否存在多项式的关系，如果是这样，那么机器学习就是可行的。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程。



林轩田《机器学习基石》课程笔记6 -- Theory of Generalization

作者：红色石头　　公众号：AI有道（id：redstonewill）

上一节课，我们主要探讨了当M的数值大小对机器学习的影响。如果M很大，那么就不能保证机器学习有很好的泛化能力，所以问题转换为验证M有限，即最好是按照多项式成长。然后通过引入了成长函数$m_H(N)$和dichotomy以及break point的概念，提出2D perceptrons的成长函数$m_H(N)$是多项式级别的猜想。这就是本节课将要深入探讨和证明的内容。

一、Restriction of Break Point

我们先回顾一下上节课的内容，四种成长函数与break point的关系：

- positive rays: $m_{\mathcal{H}}(N) = N + 1$
  ○× $m_{\mathcal{H}}(2) = 3 < 2^2$: break point at 2
- positive intervals: $m_{\mathcal{H}}(N) = \frac{1}{2}N^2 + \frac{1}{2}N + 1$
  ○×○ $m_{\mathcal{H}}(3) = 7 < 2^3$: break point at 3
- convex sets: $m_{\mathcal{H}}(N) = 2^N$
  $m_{\mathcal{H}}(N) = 2^N$ always: no break point
- 2D perceptrons: $m_{\mathcal{H}}(N) < 2^N$ in some cases
  $m_{\mathcal{H}}(4) = 14 < 2^4$: break point at 4

下面引入一个例子，如果k=2，那么当N取不同值的时候，计算其成长函数$m_H(N)$是多少。很明显，当N=1时，$m_H(N)$=2,；当N=2时，由break point为2可知，任意两点都不能被shattered（shatter的意思是对N个点，能够分解为2^N种dichotomies）；$m_H(N)$最大值只能是3；当N=3时，简单绘图分析可得其$m_H(N) = 4$，即最多只有4种dichotomies。



what 'must be true' when **minimum break point** $k = 2$

- $N = 1$: every $m_{\mathcal{H}}(N) = 2$ by definition
- $N = 2$: every $m_{\mathcal{H}}(N) < 4$ by definition (so **maximum possible = 3**)
- $N = 3$: **maximum possible = 4** $\ll 2^3$

—break point k **restricts maximum possible** $m_{\mathcal{H}}(N)$ **a lot** for $N > k$

所以，我们发现当N>k时，break point限制了$m_H(N)$值的大小，也就是说影响成长函数$m_H(N)$的因素主要有两个：

- 抽样数据集N
- break point k（这个变量确定了假设的类型）

那么，如果给定N和k，能够证明其$m_H(N)$的最大值的上界是多项式的，则根据霍夫丁不等式，就能用$m_H(N)$代替M，得到机器学习是可行的。所以，证明$m_H(N)$的上界是poly(N)，是我们的目标。

idea: $m_{\mathcal{H}}(N)$
$\leq$ maximum possible $m_{\mathcal{H}}(N)$ given k
$\leq$ $poly(N)$

二、Bounding Function: Basic Cases

现在，我们引入一个新的函数：bounding function，B(N,k)。Bound Function指的是当break point为k的时候，成长函数$m_H(N)$可能的最大值。也就是说B(N,k)是$m_H(N)$的上界，对应$m_H(N)$最多有多少种dichotomy。那么，我们新的目标就是证明：

$$B(N,k) \leq poly(N)$$

这里值得一提的是，B(N,k)的引入不考虑是1D postive intrervals问题还是2D perceptrons问题，而只关心成长函数的上界是多少，从而简化了问题的复杂度。



bounding function $B(N, k)$:
maximum possible $m_{\mathcal{H}}(N)$ when break point = k

- combinatorial quantity:
  maximum number of length-N vectors with (○, ×)
  while **'no shatter' any length-k** subvectors
- irrelevant of the details of $\mathcal{H}$
  e.g. $B(N, 3)$ bounds both
  - positive intervals ($k = 3$)
  - 1D perceptrons ($k = 3$)

求解B(N,k)的过程十分巧妙：

- 当k=1时，B(N,1)恒为1。
- 当N < k时，根据break point的定义，很容易得到$B(N,k)=2^N$。
- 当N = k时，此时N是第一次出现不能被shatter的值，所以最多只能有2^N-1个dichotomies，则$B(N,k)=2^N-1$。

$B(N,k)$	k=1	2	3	4	5	6	...
N=1	1	2	2	2	2	2	...
2	1	3	4	4	4	4	...
3	1	4	7	8	8	8	...
4	1			15	16	16	...
5	1				31	32	...
6	1					63	...
⋮	⋮						⋱

到此，bounding function的表格已经填了一半了，对于最常见的N>k的情况比较复杂，推导过程下一小节再详细介绍。

三、Bounding Function: Inductive Cases

N > k的情况较为复杂，下面给出推导过程：

以B(4,3)为例，首先想着能否构建B(4,3)与B(3,x)之间的关系。

首先，把B(4,3)所有情况写下来，共有11组。也就是说再加一种dichotomy，任意三点都能被shattered，11是极限。

	x_1	x_2	x_3	x_4
01	o	o	o	o
02	×	o	o	o
03	o	×	o	o
04	o	o	×	o
05	o	o	o	×
06	×	×	o	×
07	×	o	×	o
08	×	o	o	×
09	o	×	×	o
10	o	×	o	×
11	o	o	×	×

对这11种dichotomy分组，目前分成两组，分别是orange和purple，orange的特点是，x1,x2和x3是一致的，x4不同并成对，例如1和5，2和8等，purple则是单一的，x1,x2,x3都不同，如6,7,9三组。

	x_1	x_2	x_3	x_4
01	o	o	o	o
02	×	o	o	o
03	o	×	o	o
04	o	o	×	o
05	o	o	o	×
06	×	×	o	×
07	×	o	×	o
08	×	o	o	×
09	o	×	×	o
10	o	×	o	×
11	o	o	×	×

$\Rightarrow$

	x_1	x_2	x_3	x_4
01	o	o	o	o
05	o	o	o	×
02	×	o	o	o
08	×	o	o	×
03	o	×	o	o
10	o	×	o	×
04	o	o	×	o
11	o	o	×	×
06	×	×	o	×
07	×	o	×	o
09	o	×	×	o

orange: pair; **purple**: single

将Orange去掉x4后去重得到4个不同的vector并成为α，相应的purple为β。那么$B(4,3)=2\alpha+\beta$，这个是直接转化。紧接着，由定义，B(4,3)是不能允许任意三点shatter的，所以由α和β构成的所有三点组合也不能shatter（alpha经过去重），即$\alpha+\beta\leq B(3,3)$。



	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$
α	o	o	o
	×	o	o
	o	×	o
	o	o	×
β	×	×	o
	×	o	×
	o	×	×

- $\alpha + \beta$: dichotomies on $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$
- $B(4,3)$ 'no shatter' any 3 inputs
  $\Longrightarrow \alpha + \beta$ 'no shatter' any 3

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
2α	o	o	o	o
	o	o	o	×
	×	o	o	o
	×	o	o	×
	o	×	o	o
	o	×	o	×
	o	o	×	o
	o	o	×	×
β	×	×	o	×
	×	o	×	o
	o	×	×	o

$\alpha + \beta \leq B(3,3)$

另一方面，由于α中x4是成对存在的，且α是不能被任意三点shatter的，则能推导出α是不能被任意两点shatter的。这是因为，如果α是不能被任意两点shatter，而x4又是成对存在的，那么x1、x2、x3、x4组成的α必然能被三个点shatter。这就违背了条件的设定。这个地方的推导非常巧妙，也解释了为什么会这样分组。此处得到的结论是$\alpha \leq B(3,2)$

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$
α	o	o	o
	×	o	o
	o	×	o
	o	o	×

- α: dichotomies on $(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3)$ with $\mathbf{x}_4$ **paired**
- $B(4,3)$ 'no shatter' any 3 inputs
  $\Longrightarrow \alpha$ 'no shatter' any 2

	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$	$\mathbf{x}_4$
2α	o	o	o	o
	o	o	o	×
	×	o	o	o
	×	o	o	×
	o	×	o	o
	o	×	o	×
	o	o	×	o
	o	o	×	×
β	×	×	o	×
	×	o	×	o
	o	×	×	o

$\alpha \leq B(3,2)$

由此得出B(4,3)与B(3,x)的关系为：



$$
\begin{aligned}
B(4,3) &= 2\alpha + \beta \\
\alpha + \beta &\leq B(3,3) \\
\alpha &\leq B(3,2) \\
\Rightarrow B(4,3) &\leq B(3,3) + B(3,2)
\end{aligned}
$$

最后，推导出一般公式为：

$$
\begin{aligned}
B(N,k) &= 2\alpha + \beta \\
\alpha + \beta &\leq B(N-1,k) \\
\alpha &\leq B(N-1,k-1) \\
\Rightarrow B(N,k) &\leq B(N-1,k) + B(N-1,k-1)
\end{aligned}
$$

根据推导公式，下表给出B(N,K)值

	$B(N,k)$	k = 1	2	3	4	5	6
	1	1	2	2	2	2	2
	2	1	3	4	4	4	4
	3	1	4	7	8	8	8
N	4	1	≤ 5	11	15	16	16
	5	1	≤ 6	≤ 16	≤ 26	31	32
	6	1	≤ 7	≤ 22	≤ 42	≤ 57	63

根据递推公式，推导出B(N,K)满足下列不等式：

$$
B(N,k) \leq \underbrace{\sum_{i=0}^{k-1} \binom{N}{i}}_{\text{highest term } N^{k-1}}
$$

上述不等式的右边是最高阶为k-1的N多项式，也就是说成长函数$m_H(N)$的上界B(N,K)的上界满足多项式分布poly(N)，这就是我们想要得到的结果。

得到了$m_H(N)$的上界B(N,K)的上界满足多项式分布poly(N)后，我们回过头来看看之前介绍的几种类型它们的$m_H(N)$与break point的关系：

- positive rays: $m_{\mathcal{H}}(N) = N + 1 \leq N + 1$

  ∘× $m_{\mathcal{H}}(2) = 3 < 2^2$: break point at 2
- positive intervals: $m_{\mathcal{H}}(N) = \frac{1}{2}N^2 + \frac{1}{2}N + 1 \leq \frac{1}{2}N^2 + \frac{1}{2}N + 1$

  ∘×∘ $m_{\mathcal{H}}(3) = 7 < 2^3$: break point at 3
- 2D perceptrons: $m_{\mathcal{H}}(N) = ? \leq \frac{1}{6}N^3 + \frac{5}{6}N + 1$

  × ∘ ∘ × $m_{\mathcal{H}}(4) = 14 < 2^4$: break point at 4

我们得到的结论是，对于2D perceptrons，break point为k=4，$m_H(N)$的上界是N^{k-1}。推广一下，也就是说，如果能找到一个模型的break point，且是有限大的，那么就能推断出其成长函数$m_H(N)$有界。

四、A Pictorial Proof

我们已经知道了成长函数的上界是poly(N)的，下一步，如果能将$m_H(N)$代替M，代入到Hoffding不等式中，就能得到$E_{out} \approx E_{in}$的结论：

want:

$$\mathbb{P}\left[\exists h \in \mathcal{H} \text{ s.t. } \left|E_{\text{in}}(h) - E_{\text{out}}(h)\right| > \epsilon\right] \leq 2 \quad m_{\mathcal{H}}(\quad N) \cdot \exp\left(-2 \qquad \epsilon^2 N\right)$$

实际上并不是简单的替换就可以了，正确的表达式为：

actually, when N large enough,

$$\mathbb{P}\left[\exists h \in \mathcal{H} \text{ s.t. } \left|E_{\text{in}}(h) - E_{\text{out}}(h)\right| > \epsilon\right] \leq 2 \cdot 2 m_{\mathcal{H}}(2N) \cdot \exp\left(-2 \cdot \frac{1}{16} \epsilon^2 N\right)$$

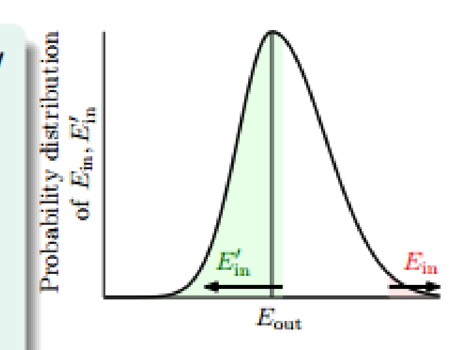
该推导的证明比较复杂，我们可以简单概括为三个步骤来证明：



Step 1: Replace E_{out} by E'_{in}

$$\frac{1}{2}\mathbb{P}\left[\exists h \in \mathcal{H} \text{ s.t. } |E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon\right]$$
$$\leq \quad \mathbb{P}\left[\exists h \in \mathcal{H} \text{ s.t. } |E_{\text{in}}(h) - E'_{\text{in}}(h)| > \frac{\epsilon}{2}\right]$$

- $E_{\text{in}}(h)$ finitely many, $E_{\text{out}}(h)$ infinitely many —replace the evil E_{out} first
- how? sample verification set $\mathcal{D}'$ of size N to calculate E'_{in}
- BAD h of $E_{\text{in}} - E_{\text{out}}$ $\overset{\text{probably}}{\Longrightarrow}$ BAD h of $E_{\text{in}} - E'_{\text{in}}$

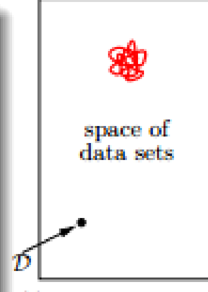


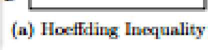
evil E_{out} removed by verification with '**ghost data**'

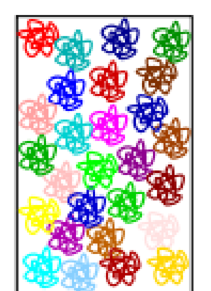
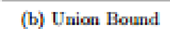
Step 2: Decompose $\mathcal{H}$ by Kind

$$\text{BAD} \quad \leq \quad 2\mathbb{P}\left[\exists h \in \mathcal{H} \text{ s.t. } |E_{\text{in}}(h) - E'_{\text{in}}(h)| > \frac{\epsilon}{2}\right]$$
$$\leq \quad 2m_{\mathcal{H}}(2N)\mathbb{P}\left[\text{fixed } h \text{ s.t. } |E_{\text{in}}(h) - E'_{\text{in}}(h)| > \frac{\epsilon}{2}\right]$$

- E_{in} with $\mathcal{D}$, E'_{in} with $\mathcal{D}'$ —now $m_{\mathcal{H}}$ comes to play
- how? infinite $\mathcal{H}$ becomes $|\mathcal{H}(\mathbf{x}_1, \ldots, \mathbf{x}_N, \mathbf{x}'_1, \ldots, \mathbf{x}'_N)|$ kinds
- union bound on $m_{\mathcal{H}}(2N)$ kinds




(a) Hoeffding Inequality



(b) Union Bound


(c) Now

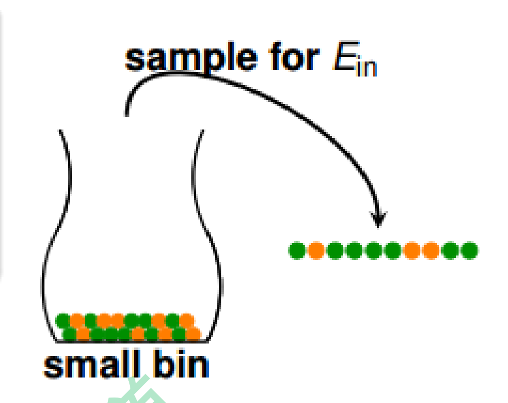
use $m_{\mathcal{H}}(2N)$ to calculate BAD-overlap properly



Step 3: Use Hoeffding without Replacement

$$\text{BAD} \leq 2m_{\mathcal{H}}(2N)\mathbb{P}\left[\text{fixed } h \text{ s.t. } \left|E_{\text{in}}(h) - E'_{\text{in}}(h)\right| > \frac{\epsilon}{2}\right]$$
$$\leq 2m_{\mathcal{H}}(2N) \cdot 2\exp\left(-2\left(\frac{\epsilon}{4}\right)^2 N\right)$$

- consider bin of $2N$ examples, choose N for E_{in}, leave others for E'_{in}
  $|E_{\text{in}} - E'_{\text{in}}| > \frac{\epsilon}{2} \Leftrightarrow \left|E_{\text{in}} - \frac{E_{\text{in}}+E'_{\text{in}}}{2}\right| > \frac{\epsilon}{4}$
- so? just 'smaller bin', 'smaller ϵ', and Hoeffding without replacement



use Hoeffding after zooming to fixed h

这部分内容，我也只能听个大概内容，对具体的证明过程有兴趣的童鞋可以自行研究一下，研究的结果记得告诉一下我哦。

最终，我们通过引入成长函数m_H，得到了一个新的不等式，称为Vapnik-Chervonenkis(VC) bound：

Vapnik-Chervonenkis (VC) bound:

$$\mathbb{P}\left[\exists h \in \mathcal{H} \text{ s.t. } \left|E_{\text{in}}(h) - E_{\text{out}}(h)\right| > \epsilon\right]$$
$$\leq 4m_{\mathcal{H}}(2N)\exp\left(-\frac{1}{8}\epsilon^2 N\right)$$

对于2D perceptrons，它的break point是4，那么成长函数$m_H(N) = O(N^3)$。所以，我们可以说2D perceptrons是可以进行机器学习的，只要找到hypothesis能让$E_{in} \approx 0$，就能保证$E_{in} \approx E_{out}$。

五、总结

本节课我们主要介绍了只要存在break point，那么其成长函数$m_H(N)$就满足poly(N)。推导过程是先引入$m_H(N)$的上界B(N,k)，B(N,k)的上界是N的k-1阶多项



式，从而得到$m_H(N)$的上界就是N的k-1阶多项式。然后，我们通过简单的三步证明，将$m_H(N)$代入了Hoffding不等式中，推导出了Vapnik-Chervonenkis(VC) bound，最终证明了只要break point存在，那么机器学习就是可行的。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程。





林轩田《机器学习基石》课程笔记7 -- The VC Dimension

作者：红色石头　　公众号：AI有道（id：redstonewill）

前几节课着重介绍了机器能够学习的条件并做了详细的推导和解释。机器能够学习必须满足两个条件：

- **假设空间H的Size M是有限的，即当N足够大的时候，那么对于假设空间中任意一个假设g，$E_{out} \approx E_{in}$。**
- **利用算法A从假设空间H中，挑选一个g，使$E_{in}(g) \approx 0$，则$E_{out} \approx 0$。**

这两个条件，正好对应着test和trian两个过程。train的目的是使损失期望$E_{in}(g) \approx 0$；test的目的是使将算法用到新的样本时的损失期望也尽可能小，即$E_{out} \approx 0$。

正因为如此，上次课引入了break point，并推导出只要break point存在，则M有上界，一定存在$E_{out} \approx E_{in}$。

本次笔记主要介绍VC Dimension的概念。同时也是总结VC Dimension与$E_{in}(g) \approx 0$，$E_{out} \approx 0$，Model Complexity Penalty（下面会讲到）的关系。

一、Definition of VC Dimension

首先，我们知道如果一个假设空间H有break point k，那么它的成长函数是有界的，它的上界称为Bound function。根据数学归纳法，Bound function也是有界的，且上界为N^{k-1}。从下面的表格可以看出，$N(k-1)$比B(N,k)松弛很多。



$$m_{\mathcal{H}}(N) \text{ of break point } k \le B(N,k) = \underbrace{\sum_{i=0}^{k-1}\binom{N}{i}}_{\text{highest term } N^{k-1}}$$

$B(N,k)$		1	2	3	4	5
	1	1	2	2	2	2
	2	1	3	4	4	4
	3	1	4	7	8	8
N	4	1	5	11	15	16
	5	1	6	16	26	31
	6	1	7	22	42	57

N^{k-1}	1	2	3	4	5
1	1	1	1	1	1
2	1	2	4	8	16
3	1	3	9	27	81
4	1	4	16	64	256
5	1	5	25	125	625
6	1	6	36	216	1296

provably & loosely, for $N \ge 2, k \ge 3$,

$$m_{\mathcal{H}}(N) \le B(N,k) = \sum_{i=0}^{k-1}\binom{N}{i} \le N^{k-1}$$

则根据上一节课的推导，VC bound就可以转换为：

For any $g = \mathcal{A}(\mathcal{D}) \in \mathcal{H}$ and 'statistical' large $\mathcal{D}$, for ~~$N \ge 2$,~~ $k \ge 3$

$$\begin{aligned} & \mathbb{P}_{\mathcal{D}}\left[\left|E_{\text{in}}(g) - E_{\text{out}}(g)\right| > \epsilon\right] \\ \le\ & \mathbb{P}_{\mathcal{D}}\left[\exists h \in \mathcal{H} \text{ s.t. } \left|E_{\text{in}}(h) - E_{\text{out}}(h)\right| > \epsilon\right] \\ \le\ & 4m_{\mathcal{H}}(2N)\exp\left(-\frac{1}{8}\epsilon^2 N\right) \\ \overset{\text{if } k \text{ exists}}{\le}\ & 4(2N)^{k-1}\exp\left(-\frac{1}{8}\epsilon^2 N\right) \end{aligned}$$

这样，不等式只与k和N相关了，一般情况下样本N足够大，所以我们只考虑k值。有如下结论：

- **若假设空间H有break point k，且N足够大，则根据VC bound理论，算法有良好的泛化能力**
- **在假设空间中选择一个矩g，使$E_{in} \approx 0$，则其在全集数据中的错误率会较低**



if ① $m_{\mathcal{H}}(N)$ breaks at k (good $\mathcal{H}$)

② N large enough (good $\mathcal{D}$)

$\Longrightarrow$ probably generalized '$E_{out} \approx E_{in}$', and

if ③ $\mathcal{A}$ picks a g with small E_{in} (good $\mathcal{A}$)

$\Longrightarrow$ probably learned! (:-) good luck)

下面介绍一个新的名词：VC Dimension。VC Dimension就是某假设集H能够shatter的最多inputs的个数，即最大完全正确的分类能力。（注意，只要存在一种分布的inputs能够正确分类也满足）。

shatter的英文意思是“粉碎”，也就是说对于inputs的所有情况都能列举出来。例如对N个输入，如果能够将2^N种情况都列出来，则称该N个输入能够被假设集H shatter。

根据之前break point的定义：假设集不能被shatter任何分布类型的inputs的最少个数。则VC Dimension等于break point的个数减一。

Definition

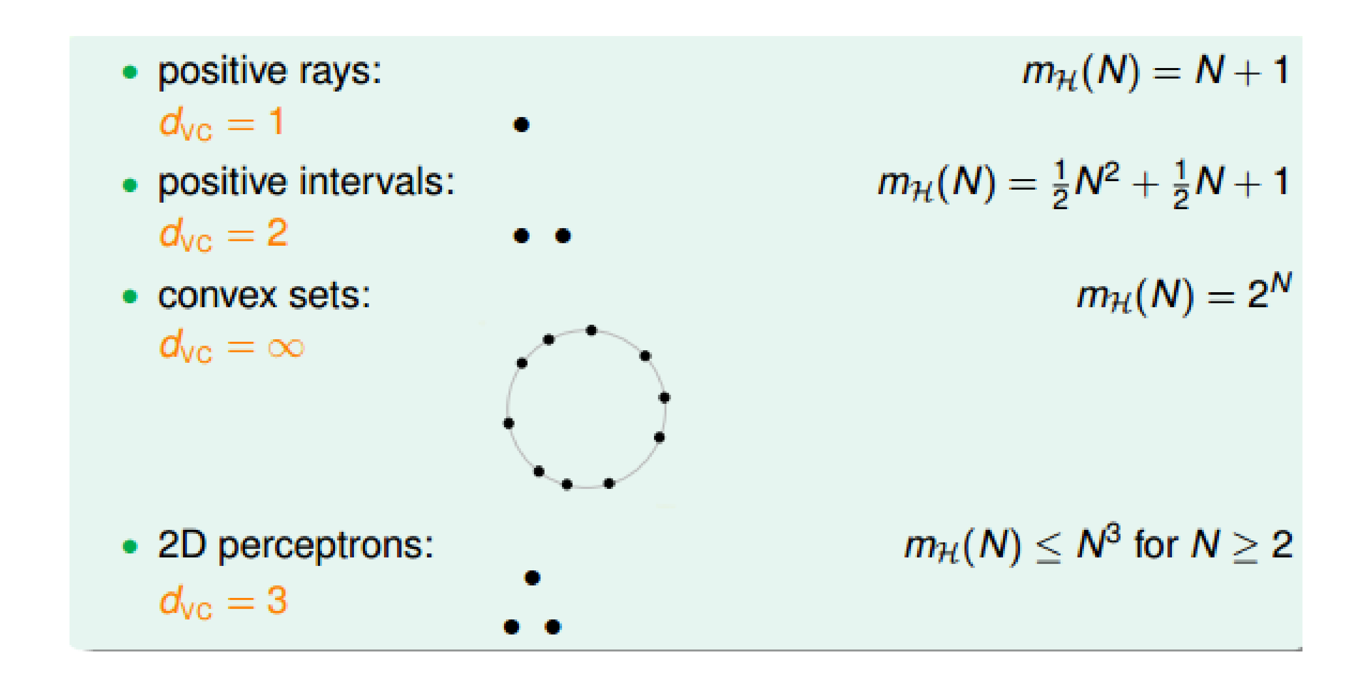
VC dimension of $\mathcal{H}$, denoted $d_{VC}(\mathcal{H})$ is

$$\text{largest } N \text{ for which } m_{\mathcal{H}}(N) = 2^N$$

- the **most** inputs $\mathcal{H}$ that can shatter
- d_{VC} = 'minimum k' - 1

现在，我们回顾一下之前介绍的四种例子，它们对应的VC Dimension是多少：

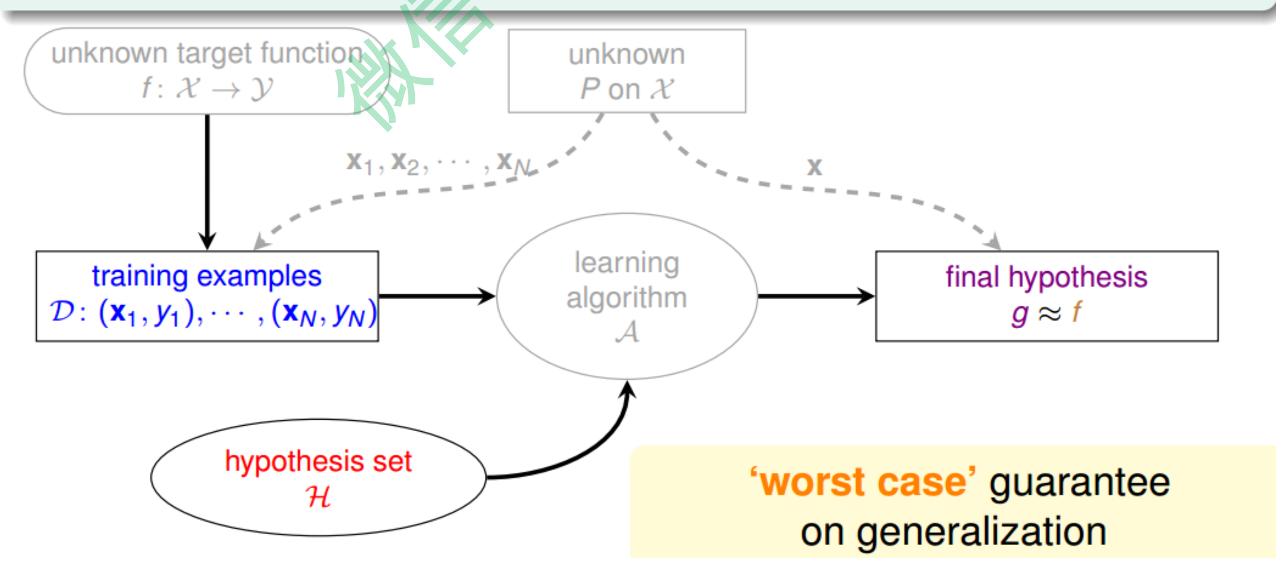




用d_{vc}代替k，那么VC bound的问题也就转换为与d_{vc}和N相关了。同时，如果一个假设集H的d_{vc}确定了，则就能满足机器能够学习的第一个条件$E_{out} \approx E_{in}$，与算法、样本数据分布和目标函数都没有关系。

finite $d_{vc} \Longrightarrow g$ 'will' generalize ($E_{out}(g) \approx E_{in}(g)$)

- regardless of learning algorithm $\mathcal{A}$
- regardless of input distribution P
- regardless of target function f

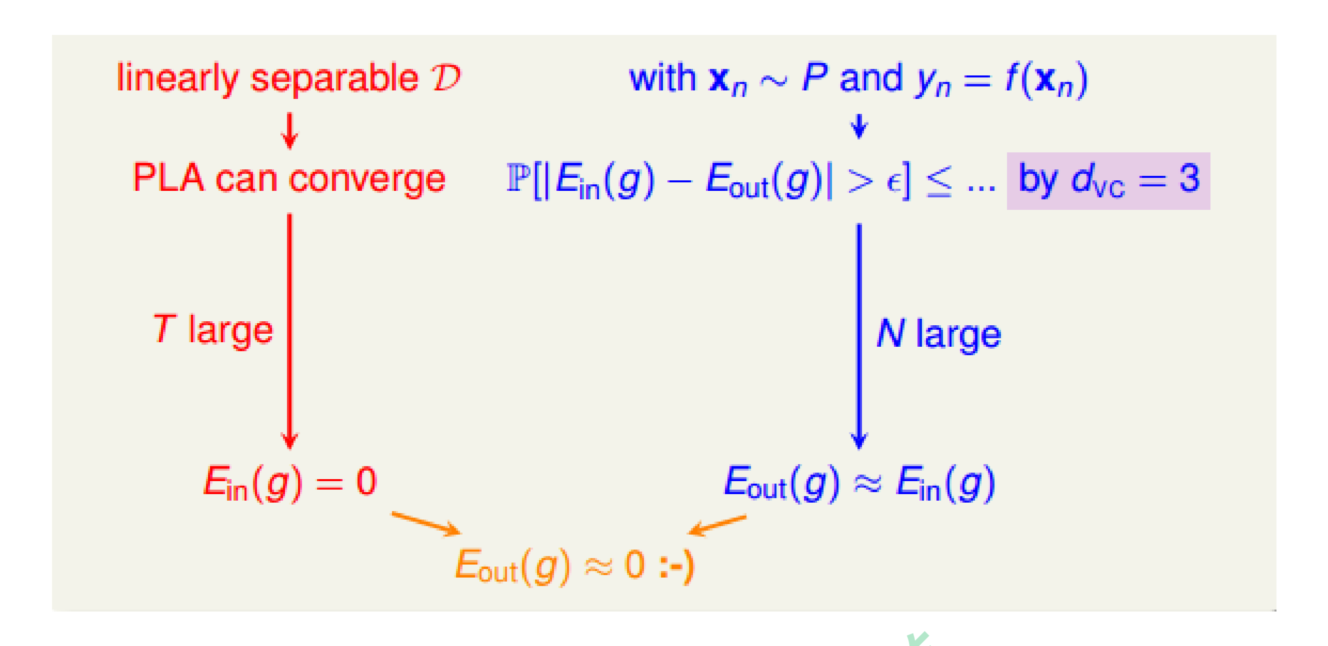


二、VC Dimension of Perceptrons

回顾一下我们之前介绍的2D下的PLA算法，已知Perceptrons的k=4，即$d_{vc} = 3$。根据VC Bound理论，当N足够大的时候，$E_{out}(g) \approx E_{in}(g)$。如果找到一个g，使



$E_{in}(g) \approx 0$，那么就能证明PLA是可以学习的。



这是在2D情况下，那如果是多维的Perceptron，它对应的d_{vc}又等于多少呢？

已知在1D Perceptron，$d_{vc} = 2$，在2D Perceptrons，$d_{vc} = 3$，那么我们有如下假设：$d_{vc} = d + 1$，其中d为维数。

要证明的话，只需分两步证明：

- $d_{vc} \geq d + 1$
- $d_{vc} \leq d + 1$

- 1D perceptron (pos/neg rays): $d_{VC} = 2$
- 2D perceptrons: $d_{VC} = 3$
  - $d_{VC} \geq 3$:
  - $d_{VC} \leq 3$:
- d-D perceptrons: $d_{VC} \stackrel{?}{=} d + 1$

首先证明第一个不等式：$d_{vc} \geq d + 1$。

在d维里，我们只要找到某一类的d+1个inputs可以被shatter的话，那么必然得到$d_{vc} \geq d + 1$。所以，我们有意构造一个d维的矩阵X能够被shatter就行。X是d维的，有d+1个inputs，每个inputs加上第零个维度的常数项1，得到X的矩阵：



$$
X = \begin{bmatrix} — x_1^T — \\ — x_2^T — \\ — x_3^T — \\ \vdots \\ — x_{d+1}^T — \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & 1 & 0 & \ldots & 0 \\ 1 & 0 & 1 & & 0 \\ \vdots & \vdots & & \ddots & 0 \\ 1 & 0 & \ldots & 0 & 1 \end{bmatrix}
$$

矩阵中，每一行代表一个inputs，每个inputs是d+1维的，共有d+1个inputs。这里构造的X很明显是可逆的。shatter的本质是假设空间H对X的所有情况的判断都是对的，即总能找到权重W，满足$X * W = y$，$W = X^{-1} * y$。由于这里我们构造的矩阵X的逆矩阵存在，那么d维的所有inputs都能被shatter，也就证明了第一个不等式。

$$
X = \begin{bmatrix} — x_1^T — \\ — x_2^T — \\ \vdots \\ — x_{d+1}^T — \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & 1 & 0 & \ldots & 0 \\ \vdots & \vdots & & \ddots & 0 \\ 1 & 0 & \ldots & 0 & 1 \end{bmatrix} \quad \text{invertible}
$$

to shatter . . .

for any $\mathbf{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_{d+1} \end{bmatrix}$, find $\mathbf{w}$ such that

$$
\text{sign}(X\mathbf{w}) = \mathbf{y} \iff (X\mathbf{w}) = \mathbf{y} \overset{X \text{ invertible!}}{\iff} \mathbf{w} = X^{-1}\mathbf{y}
$$

然后证明第二个不等式：$d_{vc} \le d + 1$。

在d维里，如果对于任何的d+2个inputs，一定不能被shatter，则不等式成立。我们构造一个任意的矩阵X，其包含d+2个inputs，该矩阵有d+1列，d+2行。这d+2个向量的某一列一定可以被另外d+1个向量线性表示，例如对于向量X_{d+2}，可表示为：

$$
X_{d+2} = a_1 * X_1 + a_2 * X_2 + \cdots + a_d * X_d
$$

其中，假设$a_1 > 0$，$a_2, \cdots, a_d < 0$.

那么如果X_1是正类，$X_2, \cdots, X_d$均为负类，则存在W，得到如下表达式：
$X_{d+2} * W = a_1 * X_1 * W + a_2 * X_2 * W + \cdots + a_d * X_d * W > 0$

因为其中蓝色项大于0，代表正类；红色项小于0，代表负类。所有对于这种情况，



X_d+2一定是正类，无法得到负类的情况。也就是说，d+2个inputs无法被shatter。证明完毕！

d-D General Case

$$X = \begin{bmatrix} -\mathbf{x}_1^T- \\ -\mathbf{x}_2^T- \\ \vdots \\ -\mathbf{x}_{d+1}^T- \\ -\mathbf{x}_{d+2}^T- \end{bmatrix}$$

more rows than columns:

linear dependence (some a_i non-zero)

$$\mathbf{x}_{d+2} = a_1\mathbf{x}_1 + a_2\mathbf{x}_2 + \ldots + a_{d+1}\mathbf{x}_{d+1}$$

- can you generate $(\text{sign}(a_1), \text{sign}(a_2), \ldots, \text{sign}(a_{d+1}), \times)$? if so, what $\mathbf{w}$?

$$\mathbf{w}^T\mathbf{x}_{d+2} = a_1 \underbrace{\mathbf{w}^T\mathbf{x}_1}_{\circ} + a_2 \underbrace{\mathbf{w}^T\mathbf{x}_2}_{\times} + \ldots + a_{d+1} \underbrace{\mathbf{w}^T\mathbf{x}_{d+1}}_{\times}$$
$$> 0 \text{(contradition!)}$$

综上证明可得$d_{vc}=d+1$。

三、Physical Intuition VC Dimension



Degrees of Freedom

(modified from the work of Hugues Vermeiren on `http://www.texample.net`)

- hypothesis parameters $\mathbf{w} = (w_0, w_1, \cdots, w_d)$:
  creates degrees of freedom
- hypothesis quantity $M = |\mathcal{H}|$:
  'analog' degrees of freedom
- hypothesis 'power' $d_{\text{VC}} = d + 1$:
  effective 'binary' degrees of freedom

$d_{\text{VC}}(\mathcal{H})$: powerfulness of $\mathcal{H}$

上节公式中W又名features，即自由度。自由度是可以任意调节的，如同上图中的旋钮一样，可以调节。VC Dimension代表了假设空间的分类能力，即反映了H的自由度，产生dichotomy的数量，也就等于features的个数，但也不是绝对的。

practical rule of thumb:

$d_{\text{VC}} \approx$ #free parameters (but not always)

例如，对2D Perceptrons，线性分类，$d_{vc} = 3$，则$W = \{w_0, w_1, w_2\}$，也就是说只要3个features就可以进行学习，自由度为3。

介绍到这，我们发现M与d_{vc}是成正比的，从而得到如下结论：



M and d_{VC}

copied from Lecture 5 :-)

1. can we make sure that $E_{out}(g)$ is close enough to $E_{in}(g)$?
2. can we make $E_{in}(g)$ small enough?

small M

1. Yes!, $\mathbb{P}[\textbf{BAD}] \le 2 \cdot M \cdot \exp(\ldots)$
2. No!, too few choices

large M

1. No!, $\mathbb{P}[\textbf{BAD}] \le 2 \cdot M \cdot \exp(\ldots)$
2. Yes!, many choices

small d_{VC}

1. Yes!, $\mathbb{P}[\textbf{BAD}] \le 4 \cdot (2N)^{d_{VC}} \cdot \exp(\ldots)$
2. No!, too limited power

large d_{VC}

1. No!, $\mathbb{P}[\textbf{BAD}] \le 4 \cdot (2N)^{d_{VC}} \cdot \exp(\ldots)$
2. Yes!, lots of power

四、Interpreting VC Dimension

下面，我们将更深入地探讨VC Dimension的意义。首先，把VC Bound重新写到这里：

For any $g = \mathcal{A}(\mathcal{D}) \in \mathcal{H}$ and 'statistical' large $\mathcal{D}$, for ~~$N \ge 2$,~~ $d_{VC} \ge 2$

$$\mathbb{P}_{\mathcal{D}}\Big[\underbrace{|E_{in}(g) - E_{out}(g)| > \epsilon}_{\textbf{BAD}}\Big] \le \underbrace{4(2N)^{d_{VC}} \exp\left(-\frac{1}{8}\epsilon^2 N\right)}_{\delta}$$

根据之前的泛化不等式，如果$|E_{in} - E_{out}| > \epsilon$，即出现bad坏的情况的概率最大不超过$\delta$。那么反过来，对于good好的情况发生的概率最小为$1 - \delta$，则对上述不等式进行重新推导：



Rephrase

..., with probability $\geq 1-\delta$, **GOOD**: $|E_{in}(g) - E_{out}(g)| \leq \epsilon$

$$\begin{aligned} \text{set} \quad \delta &= 4(2N)^{d_{VC}} \exp\left(-\frac{1}{8}\epsilon^2 N\right) \\ \frac{\delta}{4(2N)^{d_{VC}}} &= \exp\left(-\frac{1}{8}\epsilon^2 N\right) \\ \ln\left(\frac{4(2N)^{d_{VC}}}{\delta}\right) &= \frac{1}{8}\epsilon^2 N \\ \sqrt{\frac{8}{N}\ln\left(\frac{4(2N)^{d_{VC}}}{\delta}\right)} &= \epsilon \end{aligned}$$

ϵ表现了假设空间H的泛化能力，ϵ越小，泛化能力越大。

For any $g = \mathcal{A}(\mathcal{D}) \in \mathcal{H}$ and 'statistical' large $\mathcal{D}$, for ~~$N \geq 2$~~, $d_{VC} \geq 2$

$$\mathbb{P}_{\mathcal{D}}\Big[\underbrace{|E_{in}(g) - E_{out}(g)| > \epsilon}_{\textbf{BAD}}\Big] \leq \underbrace{4(2N)^{d_{VC}} \exp\left(-\frac{1}{8}\epsilon^2 N\right)}_{\delta}$$

Rephrase

..., with probability $\geq 1-\delta$, **GOOD**!

$$\text{gen. error } |E_{in}(g) - E_{out}(g)| \leq \sqrt{\frac{8}{N}\ln\left(\frac{4(2N)^{d_{VC}}}{\delta}\right)}$$

$$E_{in}(g) - \sqrt{\frac{8}{N}\ln\left(\frac{4(2N)^{d_{VC}}}{\delta}\right)} \leq E_{out}(g) \leq E_{in}(g) + \sqrt{\frac{8}{N}\ln\left(\frac{4(2N)^{d_{VC}}}{\delta}\right)}$$

$\underbrace{\sqrt{\cdots}}_{\Omega(N,\mathcal{H},\delta)}$: penalty for **model complexity**

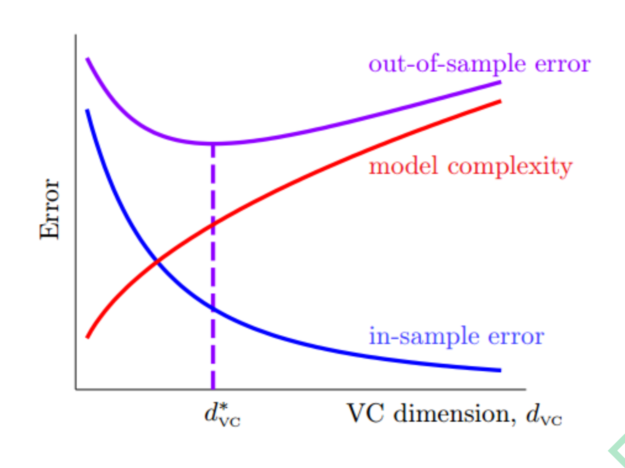
至此，已经推导出泛化误差E_{out}的边界，因为我们更关心其上界（E_{out}可能的最大值），即：



with a high probability,

$$E_{\text{out}}(g) \le E_{\text{in}}(g) + \underbrace{\sqrt{\frac{8}{N}\ln\left(\frac{4(2N)^{d_{\text{VC}}}}{\delta}\right)}}_{\Omega(N,\mathcal{H},\delta)}$$

上述不等式的右边第二项称为模型复杂度，其模型复杂度与样本数量N、假设空间H(d_{vc})、ϵ有关。E_{out}由E_{in}共同决定。下面绘出E_{out}、model complexity、E_{in}随d_{vc}变化的关系：



- $d_{\text{VC}}\uparrow$: $E_{\text{in}}\downarrow$ but $\Omega\uparrow$
- $d_{\text{VC}}\downarrow$: $\Omega\downarrow$ but $E_{\text{in}}\uparrow$
- best d^*_{VC} in the middle

powerful $\mathcal{H}$ not always good!

通过该图可以得出如下结论：

- **d_{vc}越大，E_{in}越小，Ω越大（复杂）。**
- **d_{vc}越小，E_{in}越大，Ω越小（简单）。**
- **随着d_{vc}增大，E_{out}会先减小再增大。**

所以，为了得到最小的E_{out}，不能一味地增大d_{vc}以减小E_{in}，因为E_{in}太小的时候，模型复杂度会增加，造成E_{out}变大。也就是说，选择合适的d_{vc}，选择的features个数要合适。

下面介绍一个概念：样本复杂度（Sample Complexity）。如果选定d_{vc}，样本数据D选择多少合适呢？通过下面一个例子可以帮助我们理解：



given **specs** $\epsilon = 0.1$, $\delta = 0.1$, $d_{VC} = 3$, want $4(2N)^{d_{VC}} \exp\left(-\frac{1}{8}\epsilon^2 N\right) \le \delta$

N	bound
100	2.82×10^7
1,000	9.17×10^9
10,000	1.19×10^8
100,000	1.65×10^{-38}
29,300	9.99×10^{-2}

sample complexity:
need $N \approx 10{,}000 d_{VC}$ in theory

通过计算得到N=29300，刚好满足$\delta = 0.1$的条件。N大约是d_{vc}的10000倍。这个数值太大了，实际中往往不需要这么多的样本数量，大概只需要d_{vc}的10倍就够了。N的理论值之所以这么大是因为VC Bound 过于宽松了，我们得到的是一个比实际大得多的上界。

Looseness of VC Bound

$$\mathbb{P}_{\mathcal{D}}\left[\left|E_{\text{in}}(g) - E_{\text{out}}(g)\right| > \epsilon\right] \quad \le \quad 4(2N)^{d_{VC}} \exp\left(-\frac{1}{8}\epsilon^2 N\right)$$

theory: $N \approx 10{,}000 d_{VC}$; practice: $N \approx 10 d_{VC}$

Why?

- Hoeffding for unknown E_{out} — **any distribution, any target**
- $m_{\mathcal{H}}(N)$ instead of $|\mathcal{H}(\mathbf{x}_1, \ldots, \mathbf{x}_N)|$ — **'any' data**
- $N^{d_{VC}}$ instead of $m_{\mathcal{H}}(N)$ — **'any' $\mathcal{H}$ of same d_{VC}**
- union bound on worst cases — **any choice made by $\mathcal{A}$**

—**but hardly better, and 'similarly loose for all models'**

值得一提的是，VC Bound是比较宽松的，而如何收紧它却不是那么容易，这也是机器学习的一大难题。但是，令人欣慰的一点是，VC Bound基本上对所有模型的宽松程度是基本一致的，所以，不同模型之间还是可以横向比较。从而，VC Bound宽松对机器学习的可行性还是没有太大影响。

五、总结

本节课主要介绍了VC Dimension的概念就是最大的non-break point。然后，我们得到了Perceptrons在d维度下的VC Dimension是d+1。接着，我们在物理意义上，将d_{vc}与自由度联系起来。最终得出结论d_{vc}不能过大也不能过小。选取合适的值，才能让E_{out}足够小，使假设空间H具有良好的泛化能力。



注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程





林轩田《机器学习基石》课程笔记8 -- Noise and Error

作者：红色石头　　公众号：AI有道（id：redstonewill）

上一节课，我们主要介绍了VC Dimension的概念。如果Hypotheses set的VC Dimension是有限的，且有足够多N的资料，同时能够找到一个hypothesis使它的$E_{in} \approx 0$，那么就能说明机器学习是可行的。本节课主要讲了数据集有Noise的情况下，是否能够进行机器学习，并且介绍了假设空间H下演算法A的Error估计。

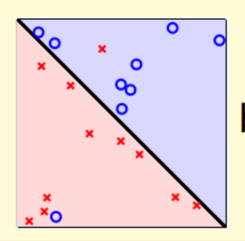
一、Noise and Probablistic target

上节课推导VC Dimension的数据集是在没有Noise的情况下，本节课讨论如果数据集本身存在Noise，那VC Dimension的推导是否还成立呢？

首先，Data Sets的Noise一般有三种情况：

- **由于人为因素，正类被误分为负类，或者负类被误分为正类；**
- **同样特征的样本被模型分为不同的类；**
- **样本的特征被错误记录和使用。**





briefly introduced **noise** before **pocket** algorithm

age	23 years
gender	female
annual salary	NTD 1,000,000
year in residence	1 year
year in job	0.5 year
current debt	200,000

credit? $\{\text{no}(-1), \text{yes}(+1)\}$

but more!

- **noise in** y: good customer, 'mislabeled' as bad?
- **noise in** y: same customers, different labels?
- **noise in x**: inaccurate customer information?

does VC bound work under **noise**?

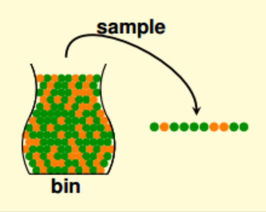
之前的数据集是确定的，即没有Noise的，我们称之为Deterministic。现在有Noise了，也就是说在某点处不再是确定分布，而是概率分布了，即对每个(x，y)出现的概率是$P(y|x)$。

因为Noise的存在，比如在x点，有0.7的概率y=1，有0.3的概率y=0，即y是按照$P(y|x)$分布的。数学上可以证明如果数据集按照$P(y|x)$概率分布且是iid的，那么以前证明机器可以学习的方法依然奏效，VC Dimension有限即可推断Ein和$Eout$是近似的。



Probabilistic Marbles

one key of VC bound: **marbles**!



'deterministic' marbles

- marble $\mathbf{x} \sim P(\mathbf{x})$
- deterministic color $[\![f(\mathbf{x}) \neq h(\mathbf{x})]\!]$

'probabilistic' (noisy) marbles

- marble $\mathbf{x} \sim P(\mathbf{x})$
- probabilistic color $[\![y \neq h(\mathbf{x})]\!]$ with $y \sim P(y|\mathbf{x})$

same nature: can estimate $\mathbb{P}$[orange] if $\overset{i.i.d.}{\sim}$

VC holds for $\underbrace{\mathbf{x} \overset{i.i.d.}{\sim} P(\mathbf{x}), y \overset{i.i.d.}{\sim} P(y|\mathbf{x})}_{(\mathbf{x},y) \overset{i.i.d.}{\sim} P(\mathbf{x},y)}$

$P(y|x)$称之为目标分布（Target Distribution）。它实际上告诉我们最好的选择是什么，同时伴随着多少noise。其实，没有noise的数据仍然可以看成“特殊”的$P(y|x)$概率分布，即概率仅是1和0.对于以前确定的数据集：

$$P(y|x) = 1, for\ y = f(x)$$

$$P(y|x) = 0, for\ y \neq f(x)$$



Target Distribution $P(y|\mathbf{x})$

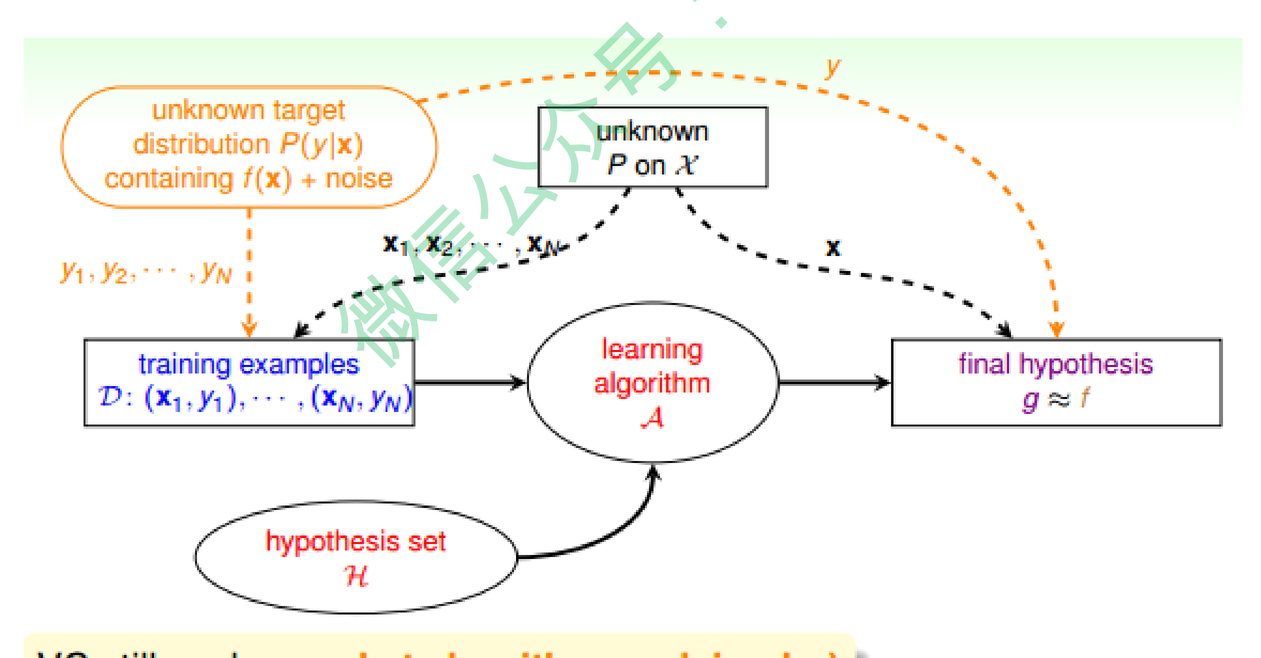
characterizes behavior of **'mini-target'** on one $\mathbf{x}$

- can be viewed as 'ideal mini-target' + noise, e.g.
  - $P(\circ|\mathbf{x}) = 0.7$, $P(\times|\mathbf{x}) = 0.3$
  - ideal mini-target $f(\mathbf{x}) = \circ$
  - 'flipping' noise level = 0.3
- deterministic target f: **special case of target distribution**
  - $P(y|\mathbf{x}) = 1$ for $y = f(\mathbf{x})$
  - $P(y|\mathbf{x}) = 0$ for $y \neq f(\mathbf{x})$

goal of learning:

predict **ideal mini-target (w.r.t. $P(y|\mathbf{x})$)**
on **often-seen inputs (w.r.t. $P(\mathbf{x})$)**

在引入noise的情况下，新的学习流程图如下所示：



VC still works, **pocket algorithm explained :-)**

二、ERROR Measure

机器学习需要考虑的问题是找出的矩g与目标函数f有多相近，我们一直使用E_{out}进行误差的估计，那一般的错误测量有哪些形式呢？

我们介绍的矩g对错误的衡量有三个特性：



- **out-of-sample：样本外的未知数据**
- **pointwise：对每个数据点x进行测试**
- **classification：看prediction与target是否一致，classification error通常称为0/1 error**

- how well? previously, considered out-of-sample measure

$$E_{out}(g) = \underset{\mathbf{x} \sim P}{\mathcal{E}} [\![g(\mathbf{x}) \neq f(\mathbf{x})]\!]$$

- more generally, **error measure** $E(g, f)$
- naturally considered
  - out-of-sample: averaged over unknown **x**
  - pointwise: evaluated on one **x**
  - classification: $[\![$prediction $\neq$ target$]\!]$

PointWise error实际上就是对数据集的每个点计算错误并计算平均，Ein和$Eout$的pointwise error的表达式为：

in-sample

$$E_{in}(g) = \frac{1}{N}\sum_{n=1}^{N} \text{err}(g(\mathbf{x}_n), f(\mathbf{x}_n))$$

out-of-sample

$$E_{out}(g) = \underset{\mathbf{x} \sim P}{\mathcal{E}} \text{err}(g(\mathbf{x}), f(\mathbf{x}))$$

pointwise error是机器学习中最常用也是最简单的一种错误衡量方式，未来课程中，我们主要考虑这种方式。pointwise error一般可以分成两类：0/1 error和squared error。0/1 error通常用在分类（classification）问题上，而squared error通常用在回归（regression）问题上。

0/1 error

$$\text{err}(\tilde{y}, y) = [\![\tilde{y} \neq y]\!]$$

- correct or incorrect?
- often for classification

squared error

$$\text{err}(\tilde{y}, y) = (\tilde{y} - y)^2$$

- how far is $\tilde{y}$ from y?
- often for regression

Ideal Mini-Target由$P(y|x)$和err共同决定，0/1 error和squared error的Ideal Mini-

Target计算方法不一样。例如下面这个例子，分别用0/1 error和squared error来估计最理想的mini-target是多少。0/1 error中的mini-target是取P(y|x)最大的那个类，而squared error中的mini-target是取所有类的加权平方和。

Ideal Mini-Target

interplay between noise and error:

$P(y|\mathbf{x})$ and err define ideal mini-target $f(\mathbf{x})$

$$P(y=1|\mathbf{x})=0.2,\ P(y=2|\mathbf{x})=0.7,\ P(y=3|\mathbf{x})=0.1$$

$$\mathrm{err}(\tilde{y},y)=[\![\tilde{y}\neq y]\!]$$

$$\tilde{y}=\begin{cases}1 & \text{avg. err } 0.8\\ 2 & \text{avg. err } 0.3(*)\\ 3 & \text{avg. err } 0.9\\ 1.9 & \text{avg. err } 1.0 \text{(really? :-))}\end{cases}$$

$$f(\mathbf{x})=\operatorname*{argmax}_{y\in\mathcal{Y}} P(y|\mathbf{x})$$

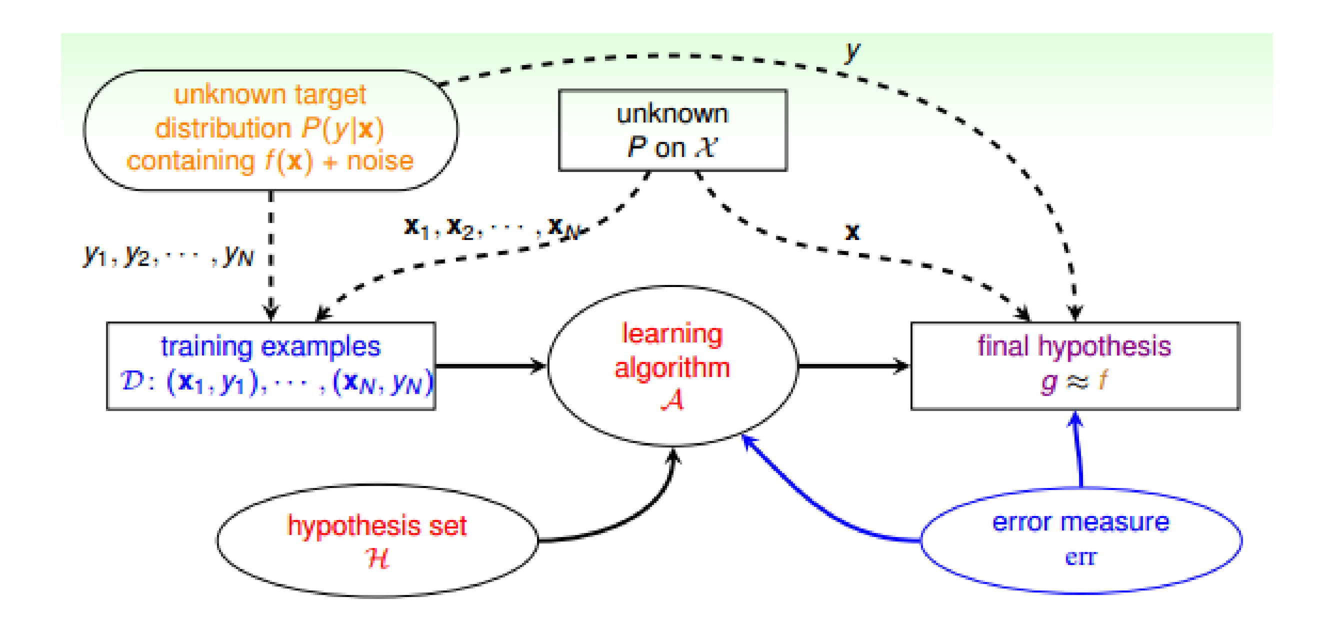
$$\mathrm{err}(\tilde{y},y)=(\tilde{y}-y)^2$$

$$\begin{cases}1 & \text{avg. err } 1.1\\ 2 & \text{avg. err } 0.3\\ 3 & \text{avg. err } 1.5\\ 1.9 & \text{avg. err } 0.29(*)\end{cases}$$

$$f(\mathbf{x})=\sum_{y\in\mathcal{Y}} y\cdot P(y|\mathbf{x})$$

有了错误衡量，就会知道当前的矩g是好还是不好，并会让演算法不断修正，得到更好的矩g，从而使得g与目标函数更接近。所以，引入error measure后，学习流程图如下所示：





三、Algorithmic Error Measure

Error有两种：false accept和false reject。false accept意思是误把负类当成正类，false reject是误把正类当成负类。 根据不同的机器学习问题，false accept和false reject应该有不同的权重，这根实际情况是符合的，比如是超市优惠，那么false reject应该设的大一些；如果是安保系统，那么false accept应该设的大一些。

two types of error: false accept and false reject

		g +1	g -1
f	+1	no error	false reject
	-1	false accept	no error

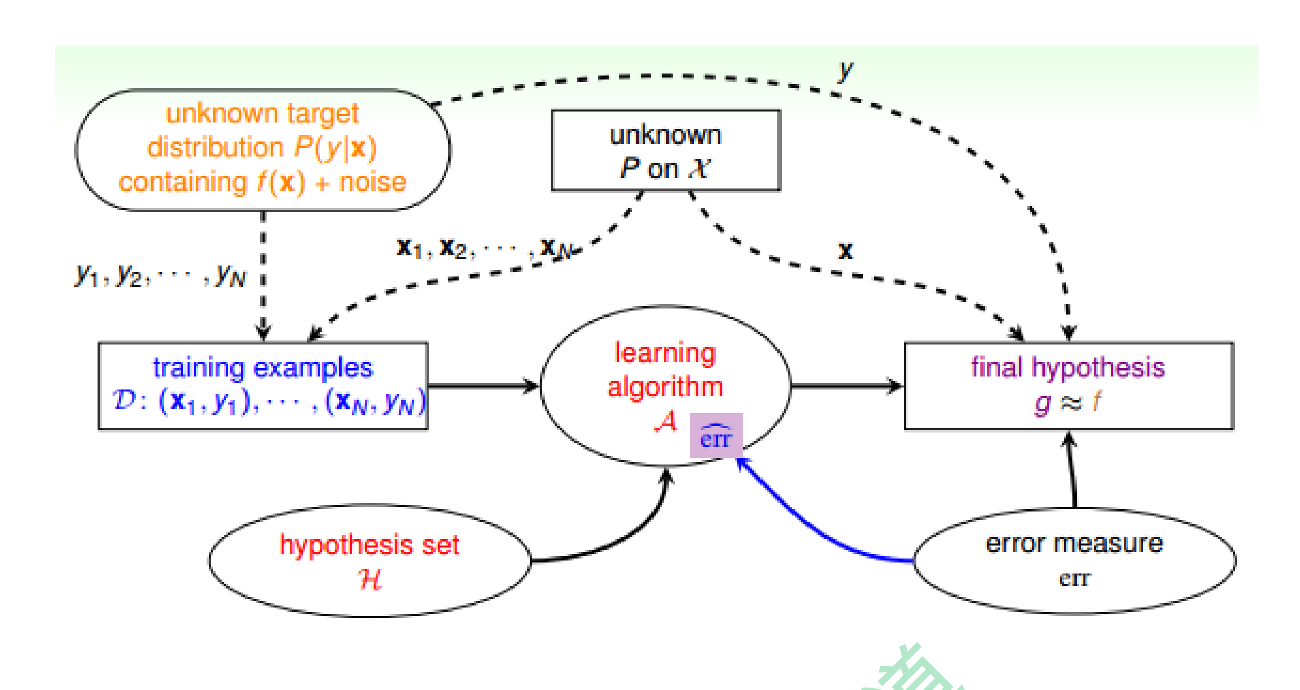
机器学习演算法A的cost function error估计有多种方法，真实的err一般难以计算，常用的方法可以采用plausible或者friendly，根据具体情况而定。

Algorithmic Error Measures $\widehat{\text{err}}$

- true: just err
- plausible:
  - 0/1: minimum 'flipping noise'—NP-hard to optimize, **remember? :-)**
  - squared: minimum Gaussian noise
- friendly: easy to optimize for $\mathcal{A}$
  - closed-form solution
  - convex objective function



引入algorithm error measure之后，学习流程图如下：



四、Weighted Classification

实际上，机器学习的Cost Function即来自于这些error，也就是算法里面的迭代的目标函数，通过优化使得Error（Ein）不断变小。

cost function中，false accept和false reject赋予不同的权重，在演算法中体现。对不同权重的错误惩罚，可以选用virtual copying的方法。



Systematic Route: Connect E_{in}^{w} and $E_{in}^{0/1}$

original problem

		$h(\mathbf{x})$ +1	$h(\mathbf{x})$ -1
y	+1	0	1
	-1	1000	0

$$\mathcal{D}: \begin{array}{c} (\mathbf{x}_1, +1) \\ (\mathbf{x}_2, -1) \\ (\mathbf{x}_3, -1) \\ \cdots \\ (\mathbf{x}_{N-1}, +1) \\ (\mathbf{x}_N, +1) \end{array}$$

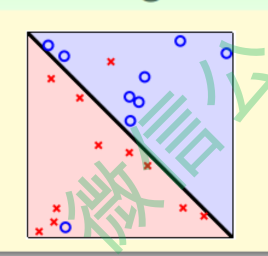
equivalent problem

		$h(\mathbf{x})$ +1	$h(\mathbf{x})$ -1
y	+1	0	1
	-1	1	0

$$\begin{array}{c} (\mathbf{x}_1, +1) \\ (\mathbf{x}_2, -1), (\mathbf{x}_2, -1), \ldots, (\mathbf{x}_2, -1) \\ (\mathbf{x}_3, -1), (\mathbf{x}_3, -1), \ldots, (\mathbf{x}_3, -1) \\ \cdots \\ (\mathbf{x}_{N-1}, +1) \\ (\mathbf{x}_N, +1) \end{array}$$

after **copying -1 examples 1000 times**,
E_{in}^{w} for LHS $\equiv E_{in}^{0/1}$ for RHS!

Weighted Pocket Algorithm



		$h(\mathbf{x})$ +1	$h(\mathbf{x})$ -1
y	+1	0	1
	-1	1000	0

using 'virtual copying', weighted pocket algorithm include:

- weighted PLA:
  randomly check -1 example mistakes with 1000 times more probability
- weighted pocket replacement:
  if $\mathbf{w}_{t+1}$ reaches smaller E_{in}^{w} than $\hat{\mathbf{w}}$, replace $\hat{\mathbf{w}}$ by $\mathbf{w}_{t+1}$

systematic route (called 'reduction'):
can be applied to many other algorithms!

五、总结



本节课主要讲了在有Noise的情况下，即数据集按照$P(y|x)$概率分布，那么VC Dimension仍然成立，机器学习算法推导仍然有效。机器学习cost function常用的Error有0/1 error和squared error两类。实际问题中，对false accept和false reject应该选择不同的权重。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程。





林轩田《机器学习基石》课程笔记9 -- Linear Regression

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课，我们主要介绍了在有noise的情况下，VC Bound理论仍然是成立的。同时，介绍了不同的error measure方法。本节课介绍机器学习最常见的一种算法：Linear Regression.

一、线性回归问题

在之前的Linear Classification课程中，讲了信用卡发放的例子，利用机器学习来决定是否给用户发放信用卡。本节课仍然引入信用卡的例子，来解决给用户发放信用卡额度的问题，这就是一个线性回归（Linear Regression）问题。

Linear Regression Hypothesis

age	23 years
annual salary	NTD 1,000,000
year in job	0.5 year
current debt	200,000

- For $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)$ 'features of customer', approximate the desired credit limit with a weighted sum:

$$y \approx \sum_{i=0}^{d} w_i x_i$$

- linear regression hypothesis: $h(\mathbf{x}) = \mathbf{w}^T\mathbf{x}$

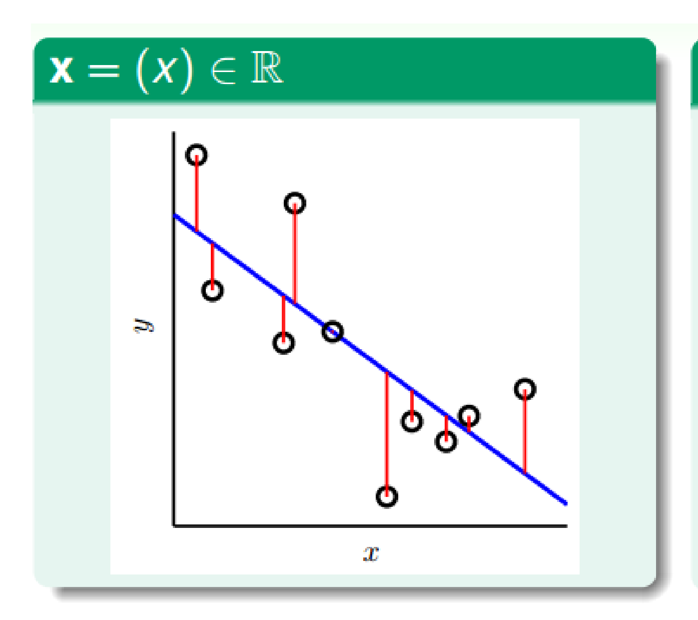
$h(\mathbf{x})$: like **perceptron**, but without the sign

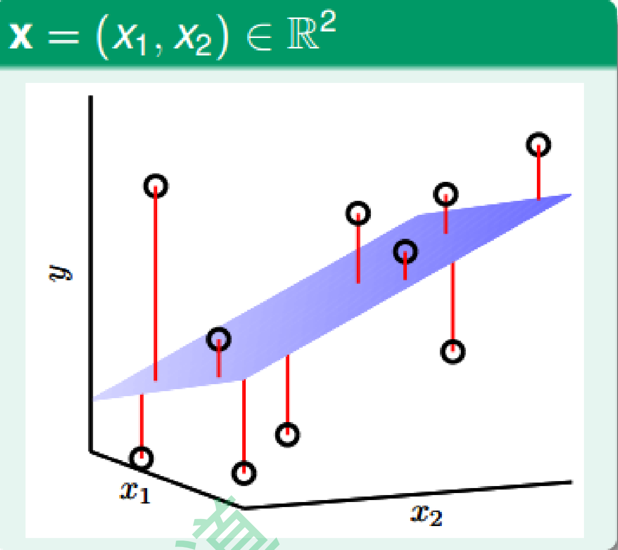
令用户特征集为d维的X，加上常数项，维度为$d+1$，与权重w的线性组合即为Hypothesis,记为$h(x)$。线性回归的预测函数取值在整个实数空间，这跟线性分类不



同。

$$h(x) = w^T X$$





linear regression:
find lines/hyperplanes with small residuals

根据上图，在一维或者多维空间里，线性回归的目标是找到一条直线（对应一维）、一个平面（对应二维）或者更高维的超平面，使样本集中的点更接近它，也就是残留误差Residuals最小化。

一般最常用的错误测量方式是基于最小二乘法，其目标是计算误差的最小平方和对应的权重w，即上节课介绍的squared error：

popular/historical error measure:

squared error $\text{err}(\hat{y}, y) = (\hat{y} - y)^2$

in-sample

$$E_{in}(h\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N}(\underbrace{h(\mathbf{x}_n)}_{\mathbf{w}^T\mathbf{x}_n} - y_n)^2$$

out-of-sample

$$E_{out}(\mathbf{w}) = \underset{(\mathbf{x},y)\sim P}{\mathcal{E}}(\mathbf{w}^T\mathbf{x} - y)^2$$

这里提一点，最小二乘法可以解决线性问题和非线性问题。线性最小二乘法的解是closed-form，即$X = (A^T A)^{-1} A^T y$，而非线性最小二乘法没有closed-form，通常



用迭代法求解。本节课的解就是closed-form的。关于最小二乘法的一些介绍，请参见我的另一篇博文：

最小二乘法和梯度下降法的一些总结

二、线性回归算法

样本数据误差E_{in}是权重w的函数，因为X和y都是已知的。我们的目标就是找出合适的w，使E_{in}能够最小。那么如何计算呢？

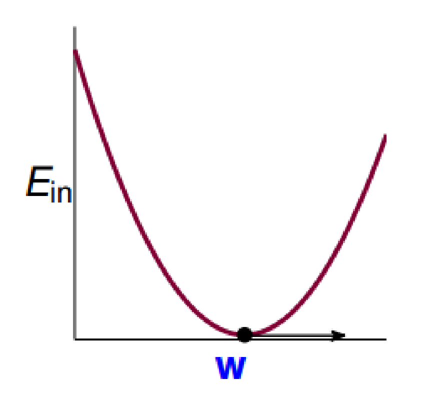
首先，运用矩阵转换的思想，将E_{in}计算转换为矩阵的形式。

$$
\begin{aligned}
E_{in}(\mathbf{w}) &= \frac{1}{N}\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{x}_n - y_n)^2 = \frac{1}{N}\sum_{n=1}^{N}(\mathbf{x}_n^T\mathbf{w} - y_n)^2 \\
&= \frac{1}{N}\left\| \begin{matrix} \mathbf{x}_1^T\mathbf{w} - y_1 \\ \mathbf{x}_2^T\mathbf{w} - y_2 \\ \cdots \\ \mathbf{x}_N^T\mathbf{w} - y_N \end{matrix} \right\|^2 \\
&= \frac{1}{N}\left\| \begin{bmatrix} --\mathbf{x}_1^T-- \\ --\mathbf{x}_2^T-- \\ \cdots \\ --\mathbf{x}_N^T-- \end{bmatrix} \mathbf{w} - \begin{bmatrix} y_1 \\ y_2 \\ \cdots \\ y_N \end{bmatrix} \right\|^2 \\
&= \frac{1}{N}\| \underbrace{\mathrm{X}}_{N\times d+1} \underbrace{\mathbf{w}}_{d+1\times 1} - \underbrace{\mathbf{y}}_{N\times 1} \|^2
\end{aligned}
$$

然后，对于此类线性回归问题，$E_{in}(w)$一般是个凸函数。凸函数的话，我们只要找到一阶导数等于零的位置，就找到了最优解。那么，我们将E_w对每个$w_i, i = 0, 1, \cdots, d$求偏导，偏导为零的w_i，即为最优化的权重值分布。



$$\min_{\mathbf{w}} E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2$$



- $E_{\text{in}}(\mathbf{w})$: continuous, differentiable, **convex**
- necessary condition of 'best' $\mathbf{w}$

$$\nabla E_{\text{in}}(\mathbf{w}) \equiv \begin{bmatrix} \frac{\partial E_{\text{in}}}{\partial w_0}(\mathbf{w}) \\ \frac{\partial E_{\text{in}}}{\partial w_1}(\mathbf{w}) \\ \dots \\ \frac{\partial E_{\text{in}}}{\partial w_d}(\mathbf{w}) \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \dots \\ 0 \end{bmatrix}$$

—not possible to 'roll down'

task: find $\mathbf{w}_{\text{LIN}}$ such that $\nabla E_{\text{in}}(\mathbf{w}_{\text{LIN}}) = \mathbf{0}$

根据梯度的思想，对E_w进行矩阵话求偏导处理：

The Gradient $\nabla E_{\text{in}}(\mathbf{w})$

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\|\mathrm{X}\mathbf{w} - \mathbf{y}\|^2 = \frac{1}{N}\Big(\mathbf{w}^T \underbrace{\mathrm{X}^T\mathrm{X}}_{\mathrm{A}} \mathbf{w} - 2\mathbf{w}^T \underbrace{\mathrm{X}^T\mathbf{y}}_{\mathbf{b}} + \underbrace{\mathbf{y}^T\mathbf{y}}_{c}\Big)$$

one w only

$$E_{\text{in}}(w) = \frac{1}{N}\left(aw^2 - 2bw + c\right)$$

$$\nabla E_{\text{in}}(w) = \frac{1}{N}(2aw - 2b)$$

simple! :-)

vector $\mathbf{w}$

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\left(\mathbf{w}^T\mathrm{A}\mathbf{w} - 2\mathbf{w}^T\mathbf{b} + c\right)$$

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{1}{N}(2\mathrm{A}\mathbf{w} - 2\mathbf{b})$$

similar (derived by definition)

$$\nabla E_{\text{in}}(\mathbf{w}) = \frac{2}{N}\left(\mathrm{X}^T\mathrm{X}\mathbf{w} - \mathrm{X}^T\mathbf{y}\right)$$

令偏导为零，最终可以计算出权重向量w为：



Optimal Linear Regression Weights

task: find $\mathbf{w}_{\text{LIN}}$ such that $\frac{2}{N}\left(X^TX\mathbf{w} - X^T\mathbf{y}\right) = \nabla E_{\text{in}}(\mathbf{w}) = \mathbf{0}$

invertible X^TX

- **easy!** unique solution

$$\mathbf{w}_{\text{LIN}} = \underbrace{\left(X^TX\right)^{-1}X^T}_{\text{pseudo-inverse } X^\dagger}\mathbf{y}$$

- often the case because $N \gg d+1$

singular X^TX

- **many** optimal solutions
- one of the solutions

$$\mathbf{w}_{\text{LIN}} = X^\dagger\mathbf{y}$$

by defining $X^\dagger$ in other ways

practical suggestion:
use **well-implemented $\dagger$ routine**
instead of $\left(X^TX\right)^{-1}X^T$
for numerical stability when **almost-singular**

最终，我们推导得到了权重向量$w=(X^TX)^{-1}X^Ty$，这是上文提到的closed-form解。其中，$(X^TX)^{-1}X^T$又称为伪逆矩阵pseudo-inverse，记为X^+，维度是(d+1)xN。

但是，我们注意到，伪逆矩阵中有逆矩阵的计算，逆矩阵$(X^TX)^{-1}$是否一定存在？一般情况下，只要满足样本数量N远大于样本特征维度d+1，就能保证矩阵的逆是存在的，称之为非奇异矩阵。但是如果是奇异矩阵，不可逆怎么办呢？其实，大部分的计算逆矩阵的软件程序，都可以处理这个问题，也会计算出一个逆矩阵。所以，一般伪逆矩阵是可解的。

三、泛化问题

现在，可能有这样一个疑问，就是这种求解权重向量的方法是机器学习吗？或者说这种方法满足我们之前推导VC Bound，即是否泛化能力强$E_{in} \approx E_{out}$？



Is Linear Regression a 'Learning Algorithm'?

$$\mathbf{w}_{\text{LIN}} = \mathrm{X}^{\dagger}\mathbf{y}$$

No!

- analytic (closed-form) solution, 'instantaneous'
- not improving E_{in} nor E_{out} iteratively

Yes!

- good E_{in}?
  yes, optimal!
- good E_{out}?
  yes, finite d_{VC} like perceptrons
- improving iteratively?
  somewhat, within an iterative pseudo-inverse routine

if $E_{\text{out}}(\mathbf{w}_{\text{LIN}})$ is good, **learning 'happened'!**

有两种观点：1、这不属于机器学习范畴。因为这种closed-form解的形式跟一般的机器学习算法不一样，而且在计算最小化误差的过程中没有用到迭代。2、这属于机器学习范畴。因为从结果上看，E_{in}和E_{out}都实现了最小化，而且实际上在计算逆矩阵的过程中，也用到了迭代。

其实，只从结果来看，这种方法的确实现了机器学习的目的。下面通过介绍一种更简单的方法，证明linear regression问题是可以通过线下最小二乘法方法计算得到好的E_{in}和E_{out}的。



Benefit of Analytic Solution: 'Simpler-than-VC' Guarantee

$$\overline{E_{in}} = \underset{\mathcal{D}\sim P^N}{\mathcal{E}}\left\{E_{in}(\mathbf{w}_{LIN} \text{ w.r.t. } \mathcal{D})\right\} \overset{\text{to be shown}}{=} \text{noise level} \cdot \left(1 - \frac{d+1}{N}\right)$$

$$E_{in}(\mathbf{w}_{LIN}) = \frac{1}{N}\|\mathbf{y} - \underbrace{\hat{\mathbf{y}}}_{\text{predictions}}\|^2 = \frac{1}{N}\|\mathbf{y} - \mathrm{X}\underbrace{\mathrm{X}^\dagger\mathbf{y}}_{\mathbf{w}_{LIN}}\|^2$$

$$= \frac{1}{N}\|(\underbrace{\mathrm{I}}_{\text{identity}} - \mathrm{X}\mathrm{X}^\dagger)\mathbf{y}\|^2$$

call $\mathrm{XX}^\dagger$ the hat matrix H
because it puts $\wedge$ on $\mathbf{y}$

首先，我们根据平均误差的思想，把$E_{in}(w_{LIN})$写成如图的形式，经过变换得到:

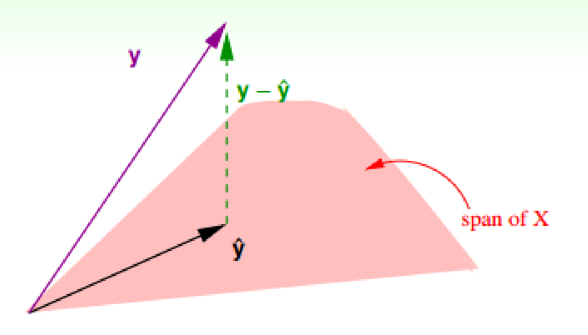
$$E_{in}(w_{LIN}) = \frac{1}{N}||(I - XX^+)y||^2 = \frac{1}{N}||(I-H)y||^2$$

我们称XX^+为帽子矩阵，用H表示。

下面从几何图形的角度来介绍帽子矩阵H的物理意义。



Geometric View of Hat Matrix



in $\mathbb{R}^N$

- $\hat{\mathbf{y}} = \mathrm{X}\mathbf{w}_{\mathrm{LIN}}$ within the span of X columns
- $\mathbf{y} - \hat{\mathbf{y}}$ smallest: $\mathbf{y} - \hat{\mathbf{y}} \perp$ span
- H: project $\mathbf{y}$ to $\hat{\mathbf{y}} \in$ span
- $\mathrm{I} - \mathrm{H}$: transform $\mathbf{y}$ to $\mathbf{y} - \hat{\mathbf{y}} \perp$ span

claim: trace(I − H) = $N - (d+1)$. Why? :-)

图中，y是N维空间的一个向量，粉色区域表示输入矩阵X乘以不同权值向量w所构成的空间，根据所有w的取值，预测输出都被限定在粉色的空间中。向量$\hat{y}$就是粉色空间中的一个向量，代表预测的一种。y是实际样本数据输出值。

机器学习的目的是在粉色空间中找到一个$\hat{y}$，使它最接近真实的y，那么我们只要将y在粉色空间上作垂直投影即可，投影得到的$\hat{y}$即为在粉色空间内最接近y的向量。这样即使平均误差$\overline{E}$最小。

从图中可以看出，$\hat{y}$是y的投影，已知$\hat{y} = Hy$，那么H表示的就是将y投影到$\hat{y}$的一种操作。图中绿色的箭头$y - \hat{y}$是向量y与$\hat{y}$相减，$y - \hat{y}$垂直于粉色区域。已知$(I - H)y = y - \hat{y}$那么I-H表示的就是将y投影到$y - \hat{y}$即垂直于粉色区域的一种操作。这样的话，我们就赋予了H和I-H不同但又有联系的物理意义。

这里trace(I-H)称为I-H的迹，值为N-(d+1)。这条性质很重要，一个矩阵的 trace等于该矩阵的所有特征值(Eigenvalues)之和。下面给出简单证明：

$$trace(I - H) = trace(I) - trace(H)$$
$$= N - trace(XX^+) = N - trace(X(X^TX)^{-1}X^T$$
$$= N - trace(X^TX(X^TX)^{-1}) = N - trace(I_{d+1}$$
$$= N - (d+1)$$

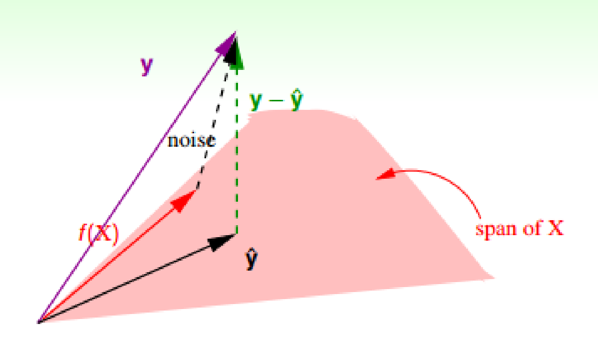
介绍下该I-H这种转换的物理意义：原来有一个有N个自由度的向量y，投影到一个有d+1维的空间x（代表一列的自由度，即单一输入样本的参数，如图中粉色区域），而



余数剩余的自由度最大只有N-(d+1)种。

在存在noise的情况下，上图变为：

An Illustrative ‘Proof’



- if **y** comes from some ideal $f(\mathrm{X}) \in$ span plus **noise**
- **noise** transformed by $\mathrm{I} - \mathrm{H}$ to be $\mathbf{y} - \hat{\mathbf{y}}$

$$E_{\text{in}}(\mathbf{w}_{\text{LIN}}) = \frac{1}{N}\|\mathbf{y} - \hat{\mathbf{y}}\|^2 = \frac{1}{N}\|(\mathrm{I} - \mathrm{H})\mathbf{noise}\|^2$$
$$= \frac{1}{N}(N - (d+1))\|\mathbf{noise}\|^2$$

$$\overline{E_{\text{in}}} = \textbf{noise level} \cdot \left(1 - \frac{d+1}{N}\right)$$
$$\overline{E_{\text{out}}} = \textbf{noise level} \cdot \left(1 + \frac{d+1}{N}\right) \text{ (complicated!)}$$

图中，粉色空间的红色箭头是目标函数f(x)，虚线箭头是noise，可见，真实样本输出y由f(x)和noise相加得到。由上面推导，已知向量y经过I-H转换为$y - \hat{y}$，而noise与y是线性变换关系，那么根据线性函数知识，我们推导出noise经过I-H也能转换为$y - \hat{y}$。则对于样本平均误差，有下列推导成立：

$$E_{in}(w_{LIN}) = \frac{1}{N}||y - \hat{y}||^2 = \frac{1}{N}||(I - H)noise||^2 = \frac{1}{N}(N - (d+1))||noise||^2$$

即

$$\overline{E}_{in} = noiselevel * (1 - \frac{d+1}{N})$$

同样，对E_{out}有如下结论：

$$\overline{E}_{out} = noiselevel * (1 + \frac{d+1}{N})$$

这个证明有点复杂，但是我们可以这样理解：$\overline{E}_{in}$与$\overline{E}_{out}$形式上只差了$\frac{(d+1)}{N}$项，从

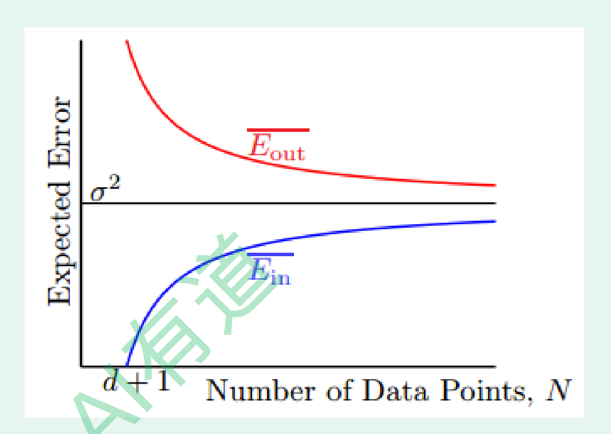


学上来说，$\overline{E}_{in}$是我们看得到的样本的平均误差，如果有noise，我们把预测往noise那边偏一点，让$\overline{E}_{in}$好看一点点，所以减去$\frac{(d+1)}{N}$项。那么同时，新的样本$\overline{E}_{out}$是我们看不到的，如果noise在反方向，那么$\overline{E}_{out}$就应该加上$\frac{(d+1)}{N}$项。

我们把$\overline{E}_{in}$与$\overline{E}_{out}$画出来，得到学习曲线：

The Learning Curve

$$\overline{E_{\text{out}}} = \textbf{noise} \text{ level} \cdot \left(1 + \frac{d+1}{N}\right)$$
$$\overline{E_{\text{in}}} = \textbf{noise} \text{ level} \cdot \left(1 - \frac{d+1}{N}\right)$$



- both converge to σ^2 (**noise** level) for $N \to \infty$
- expected generalization error: $\frac{2(d+1)}{N}$
  —**similar to worst-case guarantee from VC**

linear regression (LinReg):
learning 'happened'!

当N足够大时，$\overline{E}_{in}$与$\overline{E}_{out}$逐渐接近，满足$\overline{E}_{in} \approx \overline{E}_{out}$，且数值保持在noise level。这就类似VC理论，证明了当N足够大的时候，这种线性最小二乘法是可以进行机器学习的，算法有效！

四、Linear Regression方法解决Linear Classification问题

之前介绍的Linear Classification问题使用的Error Measure方法用的是0/1 error，那么Linear Regression的squared error是否能够应用到Linear Classification问题？



Linear Classification vs. Linear Regression

Linear Classification

$$\begin{aligned}\mathcal{Y} &= \{-1,+1\} \\ h(\mathbf{x}) &= \text{sign}(\mathbf{w}^T\mathbf{x}) \\ \text{err}(\hat{y},y) &= [\![\hat{y} \neq y]\!]\end{aligned}$$

NP-hard to solve in general

Linear Regression

$$\begin{aligned}\mathcal{Y} &= \mathbb{R} \\ h(\mathbf{x}) &= \mathbf{w}^T\mathbf{x} \\ \text{err}(\hat{y},y) &= (\hat{y}-y)^2\end{aligned}$$

efficient analytic solution

$\{-1,+1\} \subset \mathbb{R}$: linear regression for classification?

1. run LinReg on binary classification data $\mathcal{D}$ (**efficient**)
2. return $g(\mathbf{x}) = \text{sign}(\mathbf{w}_{\text{LIN}}^T\mathbf{x})$

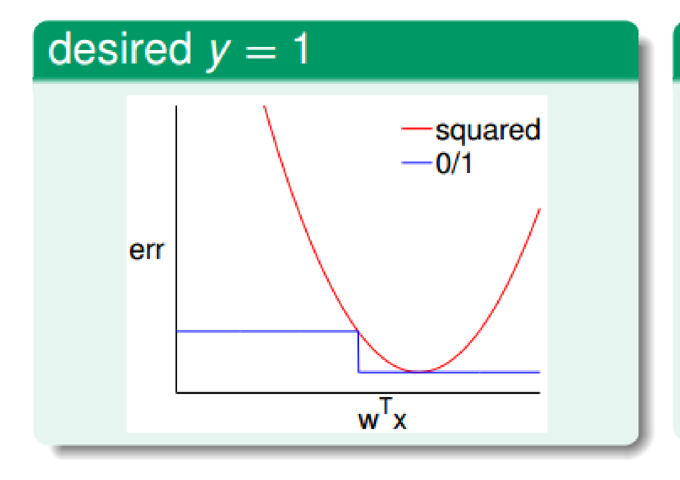
but explanation of this **heuristic**?

下图展示了两种错误的关系，一般情况下，squared error曲线在0/1 error曲线之上。即 $err_{0/1} \leq err_{sqr}$.

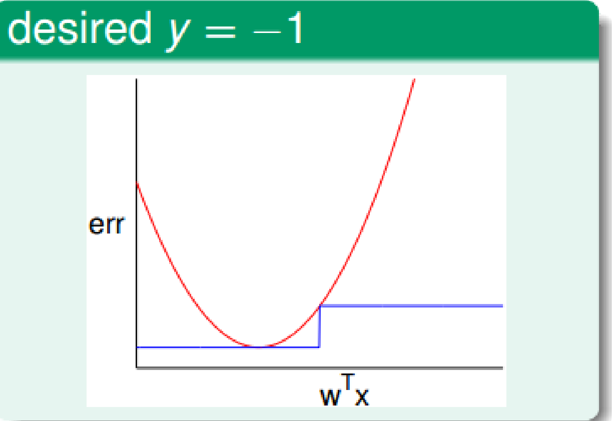
Relation of Two Errors

$$\text{err}_{0/1} = [\![\text{sign}(\mathbf{w}^T\mathbf{x}) \neq y]\!] \qquad \text{err}_{\text{sqr}} = (\mathbf{w}^T\mathbf{x} - y)^2$$

desired $y = 1$



desired $y = -1$



$$\text{err}_{0/1} \leq \text{err}_{\text{sqr}}$$

根据之前的VC理论，E_{out} 的上界满足：



Linear Regression for Binary Classification

$\text{err}_{0/1} \leq \text{err}_{\text{sqr}}$

$$\text{classification } E_{\text{out}}(\mathbf{w}) \overset{\text{VC}}{\leq} \text{classification } E_{\text{in}}(\mathbf{w}) + \sqrt{\ldots\ldots}$$
$$\leq \text{regression } E_{\text{in}}(\mathbf{w}) + \sqrt{\ldots\ldots}$$

- (loose) upper bound err_{sqr} as $\widehat{\text{err}}$ to approximate $\text{err}_{0/1}$
- trade **bound tightness** for **efficiency**

$\mathbf{w}_{\text{LIN}}$: useful baseline classifier,
or as **initial PLA/pocket vector**

从图中可以看出，用err_{sqr}代替$err_{0/1}$，E_{out}仍然有上界，只不过是上界变得宽松了。也就是说用线性回归方法仍然可以解决线性分类问题，效果不会太差。二元分类问题得到了一个更宽松的上界，但是也是一种更有效率的求解方式。

五、总结

本节课，我们主要介绍了Linear Regression。首先，我们从问题出发，想要找到一条直线拟合实际数据值；然后，我们利用最小二乘法，用解析形式推导了权重w的closed-form解；接着，用图形的形式得到$E_{out} - E_{in} \approx \frac{2(N+1)}{N}$，证明了linear regression是可以进行机器学习的，；最后，我们证明linear regressin这种方法可以用在binary classification上，虽然上界变宽松了，但是仍然能得到不错的学习方法。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程



林轩田《机器学习基石》课程笔记10 -- Logistic Regression

作者：红色石头　　公众号：AI有道（id：redstonewill）

上一节课，我们介绍了Linear Regression线性回归，以及用平方错误来寻找最佳的权重向量w，获得最好的线性预测。本节课将介绍Logistic Regression逻辑回归问题。

一、Logistic Regression Problem

一个心脏病预测的问题：根据患者的年龄、血压、体重等信息，来预测患者是否会有心脏病。很明显这是一个二分类问题，其输出y只有{-1,1}两种情况。

二元分类，一般情况下，理想的目标函数f(x)>0.5，则判断为正类1；若f(x)<0.5，则判断为负类-1。

age	40 years
gender	male
blood pressure	130/85
cholesterol level	240
weight	70

heart disease? **yes**

unknown target distribution $P(y|\mathbf{x})$ containing $f(\mathbf{x})$ + noise

training examples $\mathcal{D}: (\mathbf{x}_1, y_1), \cdots, (\mathbf{x}_N, y_N)$

learning algorithm $\mathcal{A}$ $\widehat{\text{err}}$

final hypothesis $g \approx f$

hypothesis set $\mathcal{H}$

error measure err

binary classification:

ideal $f(\mathbf{x}) = \text{sign}\left(P(+1|\mathbf{x}) - \frac{1}{2}\right) \in \{-1, +1\}$

because of classification err

但是，如果我们想知道的不是患者有没有心脏病，而是到底患者有多大的几率是心脏



病。这表示，我们更关心的是目标函数的值（分布在0,1之间），表示是正类的概率（正类表示是心脏病）。这跟我们原来讨论的二分类问题不太一样，我们把这个问题称为软性二分类问题（'soft' binary classification）。这个值越接近1，表示正类的可能性越大；越接近0，表示负类的可能性越大。

'soft' binary classification:

$$f(\mathbf{x}) = P(+1|\mathbf{x}) \in [0,1]$$

对于软性二分类问题，理想的数据是分布在[0,1]之间的具体值，但是实际中的数据只可能是0或者1，我们可以把实际中的数据看成是理想数据加上了噪声的影响。

target function $f(\mathbf{x}) = P(+1|\mathbf{x}) \in [0,1]$

ideal (noiseless) data

$$\begin{aligned}
&(\mathbf{x}_1, y_1' = 0.9 = P(+1|\mathbf{x}_1))\\
&(\mathbf{x}_2, y_2' = 0.2 = P(+1|\mathbf{x}_2))\\
&\quad\vdots\\
&(\mathbf{x}_N, y_N' = 0.6 = P(+1|\mathbf{x}_N))
\end{aligned}$$

actual (noisy) data

$$\begin{aligned}
&(\mathbf{x}_1, y_1 = \circ \sim P(y|\mathbf{x}_1))\\
&(\mathbf{x}_2, y_2 = \times \sim P(y|\mathbf{x}_2))\\
&\quad\vdots\\
&(\mathbf{x}_N, y_N = \times \sim P(y|\mathbf{x}_N))
\end{aligned}$$

same data as hard binary classification,
different **target function**

如果目标函数是$f(x) = P(+1|x) \in [0,1]$的话，我们如何找到一个好的Hypothesis跟这个目标函数很接近呢？

首先，根据我们之前的做法，对所有的特征值进行加权处理。计算的结果s，我们称之为'risk score'：



- For $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)$ 'features of patient', calculate a weighted 'risk score':

$$s = \sum_{i=0}^{d} w_i x_i$$

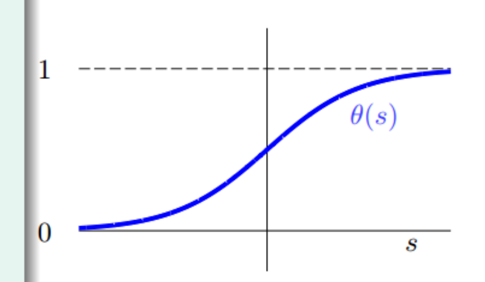
- convert the score to estimated probability by logistic function $\theta(s)$

但是特征加权和$s \in (-\infty, +\infty)$，如何将s值限定在[0,1]之间呢？一个方法是使用sigmoid Function，记为$\theta(s)$。那么我们的目标就是找到一个hypothesis：$h(x) = \theta(w^T x)$。

- For $\mathbf{x} = (x_0, x_1, x_2, \cdots, x_d)$ 'features of patient', calculate a weighted 'risk score':

$$s = \sum_{i=0}^{d} w_i x_i$$

- convert the score to estimated probability by logistic function $\theta(s)$



logistic hypothesis: $h(\mathbf{x}) = \theta(\mathbf{w}^T \mathbf{x})$

Sigmoid Function函数记为$\theta(s) = \frac{1}{1+e^{-s}}$，满足$\theta(-\infty) = 0$，$\theta(0) = \frac{1}{2}$，$\theta(+\infty) = 1$。这个函数是平滑的、单调的S型函数。则对于逻辑回归问题，hypothesis就是这样的形式：

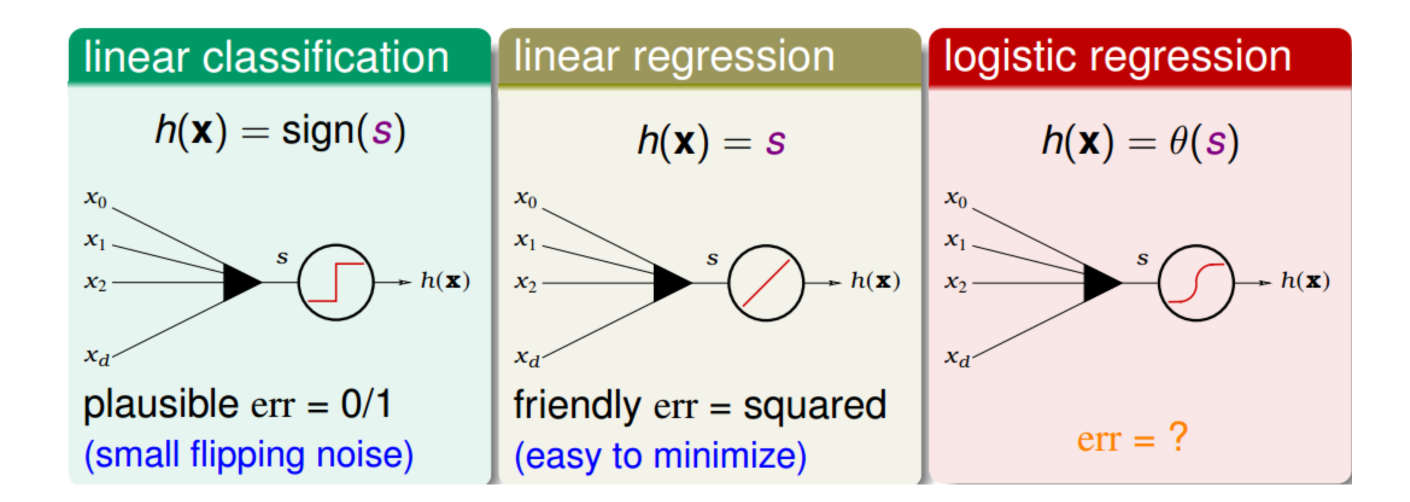
$$h(x) = \frac{1}{1 + e^{-w^T x}}$$

那我们的目标就是求出这个预测函数h(x)，使它接近目标函数f(x)。

二、Logistic Regression Error



现在我们将Logistic Regression与之前讲的Linear Classification、Linear Regression做个比较：



这三个线性模型都会用到线性scoring function $s = w^T x$。linear classification的误差使用的是0/1 err；linear regression的误差使用的是squared err。那么logistic regression的误差该如何定义呢？

先介绍一下"似然性"的概念。目标函数$f(x) = P(+1|x)$，如果我们找到了hypothesis很接近target function。也就是说，在所有的Hypothesis集合中找到一个hypothesis与target function最接近，能产生同样的数据集D，包含y输出label，则称这个hypothesis是最大似然likelihood。



Likelihood

target function $f(\mathbf{x}) = P(+1|\mathbf{x})$ $\Leftrightarrow$ $P(y|\mathbf{x}) = \begin{cases} f(\mathbf{x}) & \text{for } y = +1 \\ 1 - f(\mathbf{x}) & \text{for } y = -1 \end{cases}$

consider $\mathcal{D} = \{(\mathbf{x}_1, \circ), (\mathbf{x}_2, \times), \ldots, (\mathbf{x}_N, \times)\}$

probability that f generates $\mathcal{D}$

$$P(\mathbf{x}_1)f(\mathbf{x}_1) \times$$
$$P(\mathbf{x}_2)(1 - f(\mathbf{x}_2)) \times$$
$$\ldots$$
$$P(\mathbf{x}_N)(1 - f(\mathbf{x}_N))$$

likelihood that h generates $\mathcal{D}$

$$P(\mathbf{x}_1)h(\mathbf{x}_1) \times$$
$$P(\mathbf{x}_2)(1 - h(\mathbf{x}_2)) \times$$
$$\ldots$$
$$P(\mathbf{x}_N)(1 - h(\mathbf{x}_N))$$

- if $h \approx f$, then likelihood(h) $\approx$ probability using f
- probability using f usually **large**

logistic function: $h(x) = \theta(w^T x)$满足一个性质：$1 - h(x) = h(-x)$。那么，似然性h:

$$likelihood(h) = P(x_1)h(+x_1) \times P(x_2)h(-x_2) \times \cdots P(x_N)h(-x_N)$$

因为$P(x_n)$对所有的h来说，都是一样的，所以我们可以忽略它。那么我们可以得到logistic h正比于所有的$h(y_n x)$乘积。我们的目标就是让乘积值最大化。

$$\max_h \quad \text{likelihood(logistic } h) \propto \prod_{n=1}^{N} h(y_n \mathbf{x}_n)$$

如果将w代入的话：

$$\max_{\mathbf{w}} \quad \text{likelihood}(\mathbf{w}) \propto \prod_{n=1}^{N} \theta\left(y_n \mathbf{w}^T \mathbf{x}_n\right)$$

为了把连乘问题简化计算，我们可以引入ln操作，让连乘转化为连加：



$$\max_{\mathbf{w}} \quad \ln \prod_{n=1}^{N} \theta\left(y_n \mathbf{w}^T \mathbf{x}_n\right)$$

接着，我们将maximize问题转化为minimize问题，添加一个负号就行，并引入平均数操作$\frac{1}{N}$：

$$\min_{\mathbf{w}} \quad \frac{1}{N} \sum_{n=1}^{N} - \ln \theta\left(y_n \mathbf{w}^T \mathbf{x}_n\right)$$

将logistic function的表达式带入，那么minimize问题就会转化为如下形式：

$$\theta(s) = \frac{1}{1+\exp(-s)} \quad : \quad \min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \ln\left(1+\exp(-y_n \mathbf{w}^T \mathbf{x}_n)\right)$$

$$\Longrightarrow \min_{\mathbf{w}} \quad \underbrace{\frac{1}{N} \sum_{n=1}^{N} \mathrm{err}(\mathbf{w}, \mathbf{x}_n, y_n)}_{E_{\text{in}}(\mathbf{w})}$$

至此，我们得到了logistic regression的err function，称之为cross-entropy error交叉熵误差：

$$\mathrm{err}(\mathbf{w}, \mathbf{x}, y) = \ln\left(1+\exp(-y\mathbf{w}\mathbf{x})\right):$$

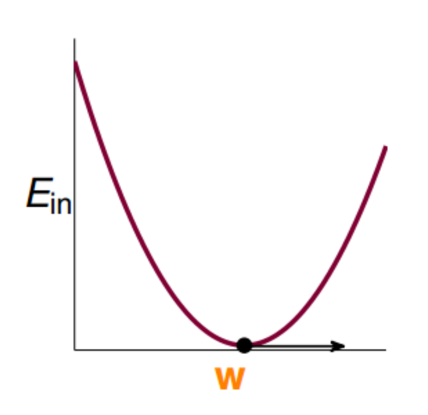
cross-entropy error

三、Gradient of Logistic Regression Error

我们已经推导了E_{in}的表达式，那接下来的问题就是如何找到合适的向量w，让E_{in}最小。

$$\min_{\mathbf{w}} \quad E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1 + \exp(-y_n \mathbf{w}^T \mathbf{x}_n)\right)$$

Logistic Regression的E_{in}是连续、可微、二次可微的凸曲线（开口向上），根据之前Linear Regression的思路，我们只要计算E_{in}的梯度为零时的w，即为最优解。



- $E_{\text{in}}(\mathbf{w})$: continuous, differentiable, twice-differentiable, **convex**
- how to minimize? locate **valley**

$$\text{want } \nabla E_{\text{in}}(\mathbf{w}) = \mathbf{0}$$

对E_{in}计算梯度，学过微积分的都应该很容易计算出来：

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\Bigg(\underbrace{1 + \exp(\overbrace{-y_n \mathbf{w}^T \mathbf{x}_n}^{\bigcirc})}_{\square}\Bigg)$$

$$\begin{aligned}
\frac{\partial E_{\text{in}}(\mathbf{w})}{\partial w_i} &= \frac{1}{N}\sum_{n=1}^{N}\left(\frac{\partial \ln(\square)}{\partial \square}\right)\left(\frac{\partial(1+\exp(\bigcirc))}{\partial \bigcirc}\right)\left(\frac{\partial - y_n \mathbf{w}^T \mathbf{x}_n}{\partial w_i}\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\left(\frac{1}{\square}\right)\left(\exp(\bigcirc)\right)\left(-y_n x_{n,i}\right) \\
&= \frac{1}{N}\sum_{n=1}^{N}\left(\frac{\exp(\bigcirc)}{1+\exp(\bigcirc)}\right)\left(-y_n x_{n,i}\right) = \frac{1}{N}\sum_{n=1}^{N}\theta(\bigcirc)\left(-y_n x_{n,i}\right)
\end{aligned}$$

最终得到的梯度表达式为：



$$\nabla E_{in}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \theta\left(-y_n\mathbf{w}^T\mathbf{x}_n\right)\left(-y_n\mathbf{x}_n\right)$$

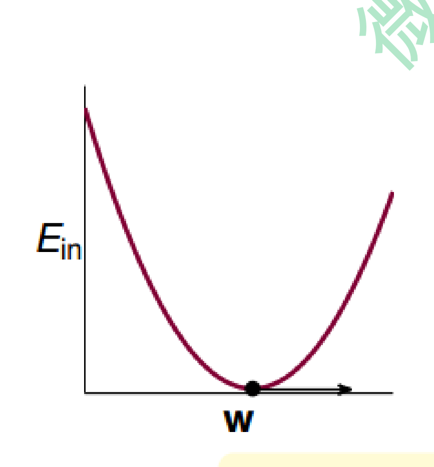
为了计算E_{in}最小值，我们就要找到让$\nabla E_{in}(w)$等于0的位置。

$$\min_{\mathbf{w}} \quad E_{in}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \ln\left(1+\exp(-y_n\mathbf{w}^T\mathbf{x}_n)\right)$$

$$\text{want } \nabla E_{in}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \theta\left(-y_n\mathbf{w}^T\mathbf{x}_n\right)\left(-y_n\mathbf{x}_n\right) = \mathbf{0}$$

上式可以看成$\theta(-y_nw^Tx_n)$是$-y_nx_n$的线性加权。要求$\theta(-y_nw^Tx_n)$与$-y_nx_n$的线性加权和为0，那么一种情况是线性可分，如果所有的权重$\theta(-y_nw^Tx_n)$为0，那就能保证$\nabla E_{in}(w)$为0。$\theta(-y_nw^Tx_n)$是sigmoid function，根据其特性，只要让$-y_nw^Tx_n \ll 0$，即$y_nw^Tx_n \gg 0$。$y_nw^Tx_n \gg 0$表示对于所有的点，y_n与w^Tx_n都是同号的，这表示数据集D必须是全部线性可分的才能成立。

然而，保证所有的权重$\theta(-y_nw^Tx_n)$为0是不太现实的，总有不等于0的时候，那么另一种常见的情况是非线性可分，只能通过使加权和为零，来求解w。这种情况没有closed-form解，与Linear Regression不同，只能用迭代方法求解。



scaled θ-weighted sum of $-y_n\mathbf{x}_n$

- all $\theta(\cdot) = 0$: only if $y_n\mathbf{w}^T\mathbf{x}_n \gg 0$ —linear separable $\mathcal{D}$
- weighted sum = **0**: non-linear equation of **w**

closed-form solution? no :-(

之前所说的Linear Regression有closed-form解，可以说是“一步登天”的；但是PLA算法是一步一步修正迭代进行的，每次对错误点进行修正，不断更新w值。PLA的迭代优化过程表示如下：



PLA: start from some $\mathbf{w}_0$ (say, $\mathbf{0}$), and 'correct' its mistakes on $\mathcal{D}$

For $t = 0, 1, \ldots$

1. find a mistake of $\mathbf{w}_t$ called $(\mathbf{x}_{n(t)}, y_{n(t)})$

$$\text{sign}\left(\mathbf{w}_t^T \mathbf{x}_{n(t)}\right) \neq y_{n(t)}$$

2. (try to) correct the mistake by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + y_{n(t)} \mathbf{x}_{n(t)}$$

1. (equivalently) pick some n, and update $\mathbf{w}_t$ by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \left[\!\left[\text{sign}\left(\mathbf{w}_t^T \mathbf{x}_n\right) \neq y_n\right]\!\right] y_n \mathbf{x}_n$$

when stop, return last $\mathbf{w}$ as g

w每次更新包含两个内容：一个是每次更新的方向$y_n x_n$，用v表示，另一个是每次更新的步长η。参数(v, η)和终止条件决定了我们的迭代优化算法。

For $t = 0, 1, \ldots$

1. (equivalently) pick some n, and update $\mathbf{w}_t$ by

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \underbrace{1}_{\eta} \cdot \underbrace{\left(\left[\!\left[\text{sign}\left(\mathbf{w}_t^T \mathbf{x}_n\right) \neq y_n\right]\!\right] \cdot y_n \mathbf{x}_n\right)}_{\mathbf{v}}$$

when stop, return last $\mathbf{w}$ as g

choice of $(\eta, \mathbf{v})$ and stopping condition defines **iterative optimization approach**

四、Gradient Descent

根据上一小节PLA的思想，迭代优化让每次w都有更新：



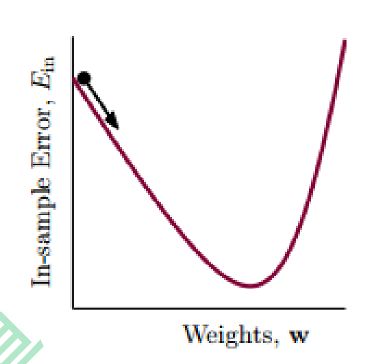
For $t = 0, 1, \ldots$

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \mathbf{v}$$

when stop, return last **w** as g

我们把$E_{in}(w)$曲线看做是一个山谷的话，要求$E_{in}(w)$最小，即可比作下山的过程。整个下山过程由两个因素影响：一个是下山的单位方向v；另外一个是下山的步长η。

- PLA: **v** comes from mistake correction
- smooth $E_{in}(\mathbf{w})$ for logistic regression: choose **v** to get the ball roll '**downhill**'?
  - direction **v**: (assumed) of unit length
  - step size η: (assumed) positive



a greedy approach for some given $\eta > 0$:

$$\min_{\|\mathbf{v}\|=1} E_{in}(\underbrace{\mathbf{w}_t + \eta \mathbf{v}}_{\mathbf{w}_{t+1}})$$

利用微分思想和线性近似，假设每次下山我们只前进一小步，即η很小，那么根据泰勒Taylor一阶展开，可以得到：

$$E_{in}(w_t + \eta v) \approx E_{in}(w_t) + \eta v^T \nabla E_{in}(w_t)$$

关于Taylor展开的介绍，可参考我另一篇博客：
多元函数的泰勒(Taylor)展开式

迭代的目的是让E_{in}越来越小，即让$E_{in}(w_t + \eta v) < E_{in}(w_t)$。$\eta$是标量，因为如果两个向量方向相反的话，那么他们的内积最小（为负），也就是说如果方向v与梯度$\nabla E_{in}(w_t)$反向的话，那么就能保证每次迭代$E_{in}(w_t + \eta v) < E_{in}(w_t)$都成立。则，我们令下降方向$v$为：

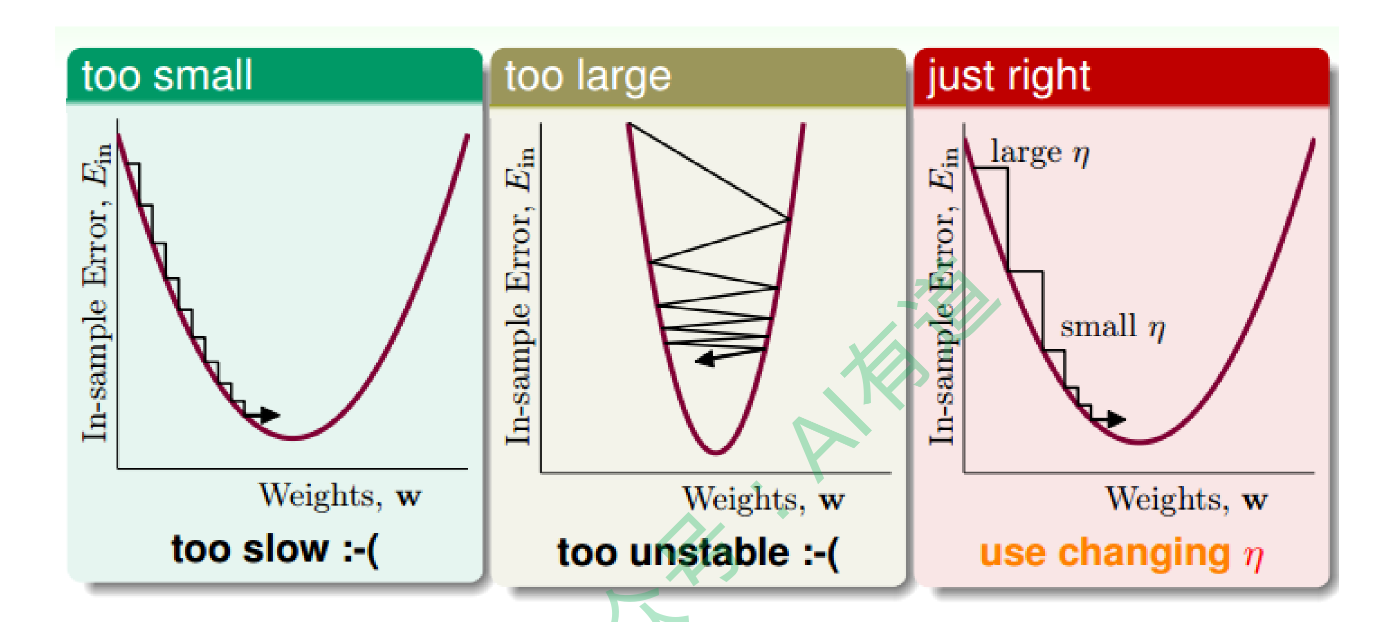
$$v = -\frac{\nabla E_{in}(w_t)}{||\nabla E_{in}(w_t)||}$$

v是单位向量，v每次都是沿着梯度的反方向走，这种方法称为梯度下降（gradient descent）算法。那么每次迭代公式就可以写成：



$$w_{t+1} \leftarrow w_t - \eta \frac{\nabla E_{in}(w_t)}{||\nabla E_{in}(w_t)||}$$

下面讨论一下η的大小对迭代优化的影响：η如果太小的话，那么下降的速度就会很慢；η如果太大的话，那么之前利用Taylor展开的方法就不准了，造成下降很不稳定，甚至会上升。因此，η应该选择合适的值，一种方法是在梯度较小的时候，选择小的η，梯度较大的时候，选择大的η，即η正比于$||\nabla E_{in}(w_t)||$。这样保证了能够快速、稳定地得到最小值$E_{in}(w)$。



对学习速率η做个更修正，梯度下降算法的迭代公式可以写成：

$$w_{t+1} \leftarrow w_t - \eta' \nabla E_{in}(w_t)$$

其中：

$$\eta' = \frac{\eta}{||\nabla E_{in}(w_t)||}$$

总结一下基于梯度下降的Logistic Regression算法步骤如下：

- **初始化w_0**
- **计算梯度$\nabla E_{in}(w_t) = \frac{1}{N}\sum_{n=1}^{N} \theta(-y_n w_t^T x_n)(-y_n x_n)$**
- **迭代跟新$w_{t+1} \leftarrow w_t - \eta \nabla E_{in}(w_t)$**
- **满足$\nabla E_{in}(w_{t+1}) \approx 0$或者达到迭代次数，迭代结束**



五、总结

我们今天介绍了Logistic Regression。首先，从逻辑回归的问题出发，将$P(+1|x)$作为目标函数，将$\theta(w^Tx)$作为hypothesis。接着，我们定义了logistic regression的err function，称之为cross-entropy error交叉熵误差。然后，我们计算logistic regression error的梯度，最后，通过梯度下降算法，计算$\nabla E_{in}(w_t) \approx 0$时对应的$w_t$值。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程





林轩田《机器学习基石》课程笔记11 -- Linear Models for Classification

作者：红色石头　　公众号：AI有道（id：redstonewill）

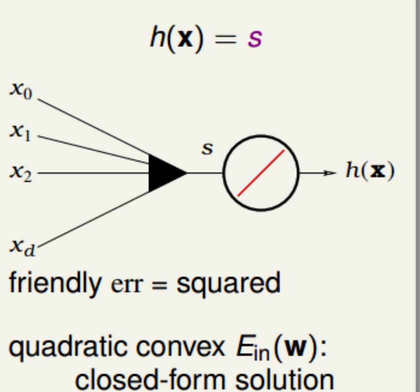
上一节课，我们介绍了Logistic Regression问题，建立cross-entropy error，并提出使用梯度下降算法gradient descnt来获得最好的logistic hypothesis。本节课继续介绍使用线性模型来解决分类问题。

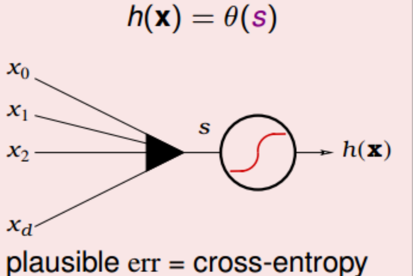
一、Linear Models for Binary Classification

之前介绍几种线性模型都有一个共同点，就是都有样本特征x的加权运算，我们引入一个线性得分函数s：

$$s = w^T x$$

三种线性模型，第一种是linear classification。线性分类模型的hypothesis为$h(x) = sign(s)$,取值范围为{-1,+1}两个值，它的err是0/1的，所以对应的$E_{in}(w)$是离散的，并不好解，这是个NP-hard问题。第二种是linear regression。线性回归模型的hypothesis为$h(x) = s$，取值范围为整个实数空间，它的err是squared的，所以对应的$E_{in}(w)$是开口向上的二次曲线，其解是closed-form的，直接用线性最小二乘法求解即可。第三种是logistic regression。逻辑回归模型的hypothesis为$h(x) = \theta(s)$，取值范围为(-1,1)之间，它的err是cross-entropy的，所有对应的$E_{in}(w)$是平滑的凸函数，可以使用梯度下降算法求最小值。







从上图中，我们发现，linear regression和logistic regression的error function都有最小解。那么可不可以用这两种方法来求解linear classification问题呢？下面，我们来对这三种模型的error function进行分析，看看它们之间有什么联系。

对于linear classification，它的error function可以写成：

$$err_{0/1}(s,y)=|sign(s)\neq y|=|sign(ys)\neq 1|$$

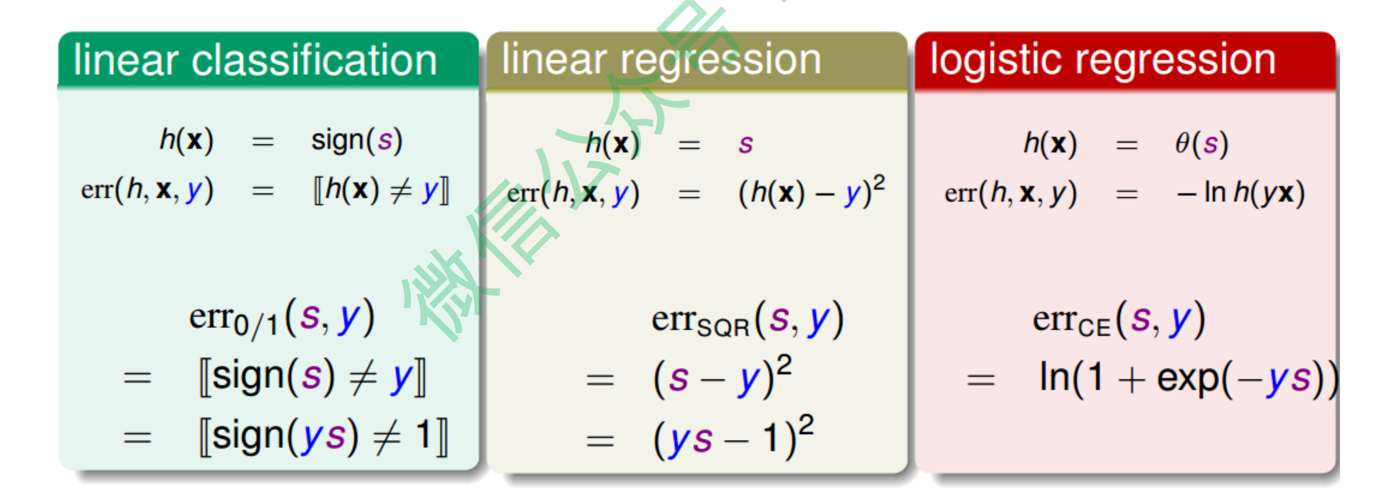
对于linear regression，它的error function可以写成：

$$err_{SQR}(s,y)=(s-y)^2=(ys-1)^2$$

对于logistic regression，它的error function可以写成：

$$err_{CE}(s,y)=ln(1+exp(-ys))$$

上述三种模型的error function都引入了ys变量，那么ys的物理意义是什么？ys就是指分类的正确率得分，其值越大越好，得分越高。

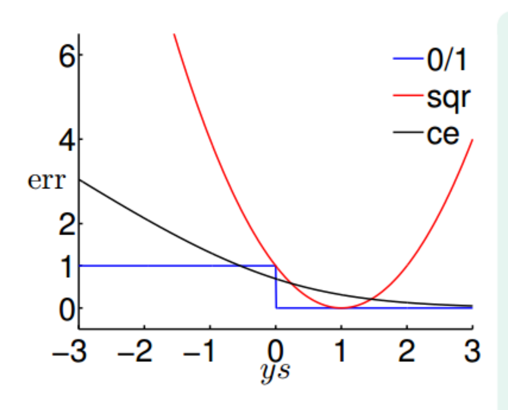


下面，我们用图形化的方式来解释三种模型的error function到底有什么关系：



Visualizing Error Functions

0/1	$\text{err}_{0/1}(s, y)$	=	$[\![\text{sign}(ys) \neq 1]\!]$
sqr	$\text{err}_{\text{SQR}}(s, y)$	=	$(ys - 1)^2$
ce	$\text{err}_{\text{CE}}(s, y)$	=	$\ln(1 + \exp(-ys))$
scaled ce	$\text{err}_{\text{SCE}}(s, y)$	=	$\log_2(1 + \exp(-ys))$

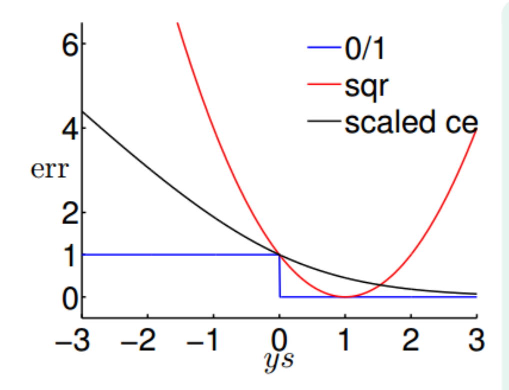


- 0/1: 1 iff $ys \leq 0$
- sqr: large if $ys \ll 1$
  but over-charge $ys \gg 1$
  small err_{SQR} → small $\text{err}_{0/1}$
- ce: monotonic of ys
  small err_{CE} ↔ small $\text{err}_{0/1}$
- scaled ce: a proper upper bound of 0/1
  small err_{SCE} ↔ small $\text{err}_{0/1}$

从上图中可以看出，ys是横坐标轴，$err_{0/1}$是呈阶梯状的，在ys>0时，$err_{0/1}$恒取最小值0。err_{SQR}呈抛物线形式，在ys=1时，取得最小值，且在ys=1左右很小区域内，$err_{0/1}$和err_{SQR}近似。err_{CE}是呈指数下降的单调函数，ys越大，其值越小。同样在ys=1左右很小区域内，$err_{0/1}$和err_{CE}近似。但是我们发现err_{CE}并不是始终在$err_{0/1}$之上，所以为了计算讨论方便，我们把err_{CE}做幅值上的调整，引入$err_{SCE} = log_2(1 + exp(-ys)) = \frac{1}{ln2}err_{CE}$，这样能保证$err_{SCE}$始终在$err_{0/1}$上面，如下图所示：

Visualizing Error Functions

0/1	$\text{err}_{0/1}(s, y)$	=	$[\![\text{sign}(ys) \neq 1]\!]$
sqr	$\text{err}_{\text{SQR}}(s, y)$	=	$(ys - 1)^2$
ce	$\text{err}_{\text{CE}}(s, y)$	=	$\ln(1 + \exp(-ys))$
scaled ce	$\text{err}_{\text{SCE}}(s, y)$	=	$\log_2(1 + \exp(-ys))$



- 0/1: 1 iff $ys \leq 0$
- sqr: large if $ys \ll 1$
  but over-charge $ys \gg 1$
  small err_{SQR} → small $\text{err}_{0/1}$
- ce: monotonic of ys
  small err_{CE} ↔ small $\text{err}_{0/1}$
- scaled ce: a proper upper bound of 0/1
  small err_{SCE} ↔ small $\text{err}_{0/1}$



由上图可以看出：

$$err_{0/1}(s,y) \le err_{SCE}(s,y) = \frac{1}{ln2} err_{CE}(s,y)$$

$$E_{in}^{0/1}(w) \le E_{in}^{SCE}(w) = \frac{1}{ln2} E_{in}^{CE}(w)$$

$$E_{out}^{0/1}(w) \le E_{out}^{SCE}(w) = \frac{1}{ln2} E_{out}^{CE}(w)$$

那么由VC理论可以知道：

从0/1出发：

$$E_{out}^{0/1}(w) \le E_{in}^{0/1}(w) + \Omega^{0/1} \le \frac{1}{ln2} E_{in}^{CE}(w) + \Omega^{0/1}$$

从CE出发：

$$E_{out}^{0/1}(w) \le \frac{1}{ln2} E_{out}^{CE}(w) \le \frac{1}{ln2} E_{in}^{CE}(w) + \frac{1}{ln2}\Omega^{CE}$$

For any ys where $s = \mathbf{w}^T\mathbf{x}$

$$\text{err}_{0/1}(s,y) \le \text{err}_{\text{SCE}}(s,y) = \frac{1}{\ln 2}\text{err}_{\text{CE}}(s,y).$$

$$\Longrightarrow \quad E_{\text{in}}^{0/1}(\mathbf{w}) \le E_{\text{in}}^{\text{SCE}}(\mathbf{w}) = \frac{1}{\ln 2} E_{\text{in}}^{\text{CE}}(\mathbf{w})$$

$$E_{\text{out}}^{0/1}(\mathbf{w}) \le E_{\text{out}}^{\text{SCE}}(\mathbf{w}) = \frac{1}{\ln 2} E_{\text{out}}^{\text{CE}}(\mathbf{w})$$

VC on $0/1$:

$$E_{\text{out}}^{0/1}(\mathbf{w}) \le E_{\text{in}}^{0/1}(\mathbf{w}) + \Omega^{0/1} \le \frac{1}{\ln 2} E_{\text{in}}^{\text{CE}}(\mathbf{w}) + \Omega^{0/1}$$

VC-Reg on CE :

$$E_{\text{out}}^{0/1}(\mathbf{w}) \le \frac{1}{\ln 2} E_{\text{out}}^{\text{CE}}(\mathbf{w}) \le \frac{1}{\ln 2} E_{\text{in}}^{\text{CE}}(\mathbf{w}) + \frac{1}{\ln 2}\Omega^{\text{CE}}$$

small $E_{\text{in}}^{\text{CE}}(\mathbf{w}) \Longrightarrow$ small $E_{\text{out}}^{0/1}(\mathbf{w})$:
logistic/linear reg. for linear classification



通过上面的分析，我们看到err 0/1是被限定在一个上界中。这个上界是由logistic regression模型的error function决定的。而linear regression其实也是linear classification的一个upper bound，只是随着sy偏离1的位置越来越远，linear regression的error function偏差越来越大。综上所述，linear regression和logistic regression都可以用来解决linear classification的问题。

下图列举了PLA、linear regression、logistic regression模型用来解linear classification问题的优点和缺点。通常，我们使用linear regression来获得初始化的w_0，再用logistic regression模型进行最优化解。

PLA	linear regression	logistic regression
• pros: **efficient + strong guarantee if lin. separable** • cons: works only if lin. separable, otherwise needing **pocket** heuristic	• pros: **'easiest' optimization** • cons: loose bound of $\mathrm{err}_{0/1}$ for large $\lvert ys \rvert$	• pros: **'easy' optimization** • cons: loose bound of $\mathrm{err}_{0/1}$ for very negative ys

- **linear regression** sometimes used to set $\mathbf{w}_0$ for **PLA/pocket/logistic regression**
- **logistic regression** often preferred over **pocket**

二、Stochastic Gradient Descent

之前介绍的PLA算法和logistic regression算法，都是用到了迭代操作。PLA每次迭代只会更新一个点，它每次迭代的时间复杂度是O(1)；而logistic regression每次迭代要对所有N个点都进行计算，它的每时间复杂度是O(N)。为了提高logistic regression中gradient descent算法的速度，可以使用另一种算法：随机梯度下降算法(Stochastic Gradient Descent)。

随机梯度下降算法每次迭代只找到一个点，计算该点的梯度，作为我们下一步更新w的依据。这样就保证了每次迭代的计算量大大减小，我们可以把整体的梯度看成这个随机过程的一个期望值。



$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \underbrace{\frac{1}{N}\sum_{n=1}^{N}\theta\left(-y_n\mathbf{w}_t^T\mathbf{x}_n\right)\left(y_n\mathbf{x}_n\right)}_{-\nabla E_{\text{in}}(\mathbf{w}_t)}$$

- want: update direction $\mathbf{v} \approx -\nabla E_{\text{in}}(\mathbf{w}_t)$
  while computing $\mathbf{v}$ by one single $(\mathbf{x}_n, y_n)$
- technique on removing $\frac{1}{N}\sum_{n=1}^{N}$:
  view as expectation $\mathcal{E}$ over uniform choice of n!

stochastic gradient:
$\nabla_{\mathbf{w}}\ \text{err}(\mathbf{w}, \mathbf{x}_n, y_n)$ with random n
true gradient:
$\nabla_{\mathbf{w}}\ E_{\text{in}}(\mathbf{w}) = \underset{\text{random } n}{\mathcal{E}}\ \nabla_{\mathbf{w}}\ \text{err}(\mathbf{w}, \mathbf{x}_n, y_n)$

随机梯度下降可以看成是真实的梯度加上均值为零的随机噪声方向。单次迭代看，好像会对每一步找到正确梯度方向有影响，但是整体期望值上看，与真实梯度的方向没有差太多，同样能找到最小值位置。随机梯度下降的优点是减少计算量，提高运算速度，而且便于online学习；缺点是不够稳定，每次迭代并不能保证按照正确的方向前进，而且达到最小值需要迭代的次数比梯度下降算法一般要多。

Stochastic Gradient Descent

- idea: replace true gradient by stochastic gradient
- after enough steps,
  average true gradient ≈ average stochastic gradient
- pros: **simple & cheaper computation :-)**
  —useful for **big data** or **online learning**
- cons: less stable in nature

对于logistic regression的SGD，它的表达式为：

$$w_{t+1} \leftarrow w_t + \eta\theta(-y_n w_t^T x_n)(y_n x_n)$$

我们发现，SGD与PLA的迭代公式有类似的地方，如下图所示：



SGD logistic regression:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + \eta \cdot \theta\left(-y_n \mathbf{w}_t^T \mathbf{x}_n\right)\left(y_n \mathbf{x}_n\right)$$

PLA:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t + 1 \cdot \left[\!\left[y_n \neq \text{sign}(\mathbf{w}_t^T \mathbf{x}_n) \right]\!\right] \left(y_n \mathbf{x}_n\right)$$

我们把SGD logistic regression称之为'soft' PLA，因为PLA只对分类错误的点进行修正，而SGD logistic regression每次迭代都会进行或多或少的修正。另外，当$\eta = 1$，且$w_t^T x_n$足够大的时候，PLA近似等于SGD。

- SGD logistic regression ≈ 'soft' PLA
- PLA ≈ SGD logistic regression with $\eta = 1$ when $\mathbf{w}_t^T \mathbf{x}_n$ large

除此之外，还有两点需要说明：1、SGD的终止迭代条件。没有统一的终止条件，一般让迭代次数足够多；2、学习速率η。η的取值是根据实际情况来定的，一般取值0.1就可以了。

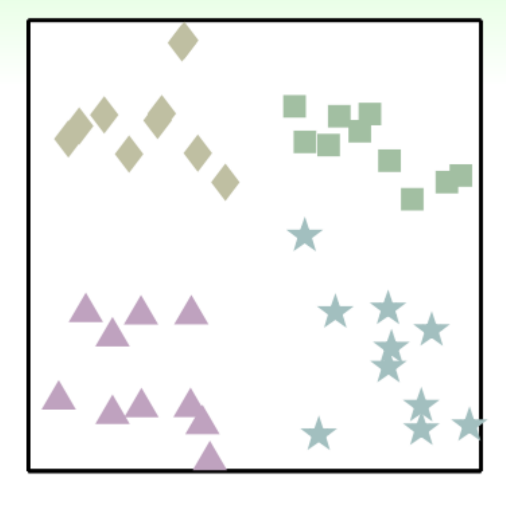
三、Multiclass via Logistic Regression

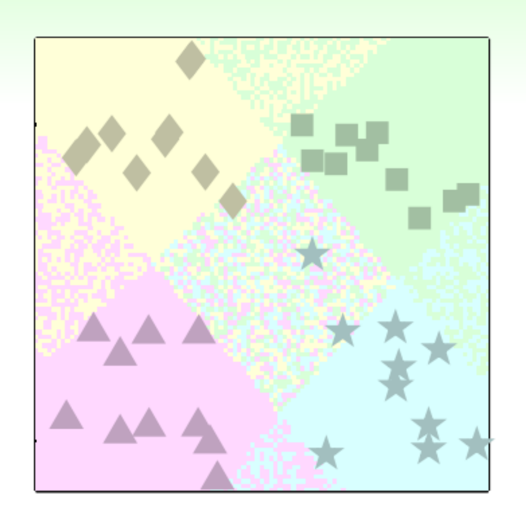
之前我们一直讲的都是二分类问题，本节主要介绍多分类问题，通过linear classification来解决。假设平面上有四个类，分别是正方形、菱形、三角形和星形，如何进行分类模型的训练呢？

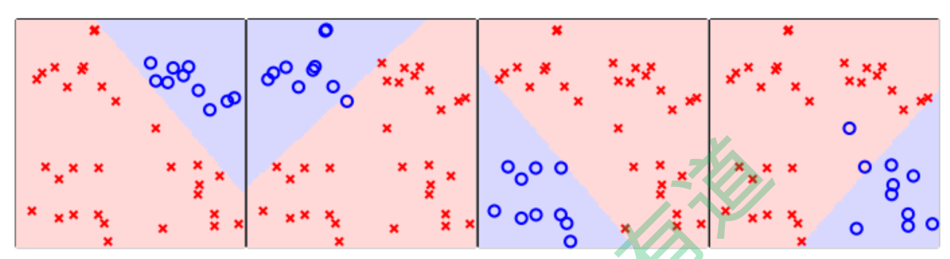
首先我们可以想到这样一个办法，就是先把正方形作为正类，其他三种形状都是负类，即把它当成一个二分类问题，通过linear classification模型进行训练，得出平面上某个图形是不是正方形，且只有{-1,+1}两种情况。然后再分别以菱形、三角形、星形为正类，进行二元分类。这样进行四次二分类之后，就完成了这个多分类问题。



Multiclass Prediction: Combine Binary Classifiers







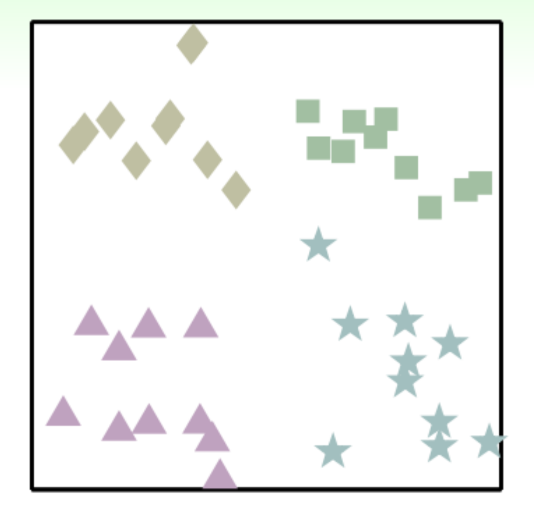
但是，这样的二分类会带来一些问题，因为我们只用{-1，+1}两个值来标记，那么平面上某些可能某些区域都被上述四次二分类模型判断为负类，即不属于四类中的任何一类；也可能会出现某些区域同时被两个类甚至多个类同时判断为正类，比如某个区域又判定为正方形又判定为菱形。那么对于这种情况，我们就无法进行多类别的准确判断，所以对于多类别，简单的binary classification不能解决问题。

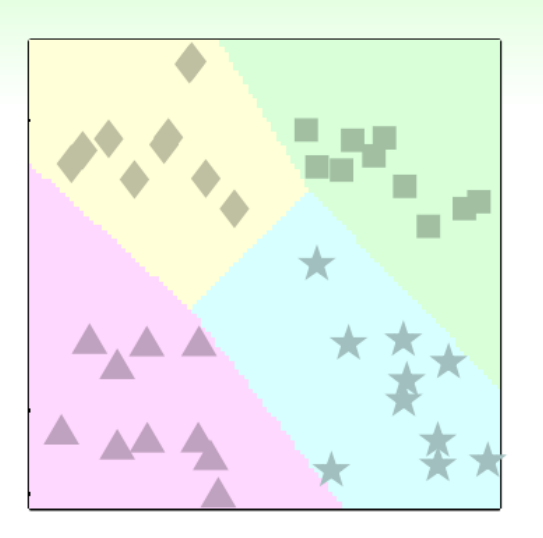
针对这种问题，我们可以使用另外一种方法来解决：soft软性分类，即不用{-1，+1}这种binary classification，而是使用logistic regression，计算某点属于某类的概率、可能性，去概率最大的值为那一类就好。

soft classification的处理过程和之前类似，同样是分别令某类为正，其他三类为负，不同的是得到的是概率值，而不是{-1，+1}。最后得到某点分别属于四类的概率，取最大概率对应的哪一个类别就好。效果如下图所示：



Multiclass Prediction: Combine Soft Classifiers





$$g(\mathbf{x}) = \text{argmax}_{k \in \mathcal{Y}}\, \theta\left(\mathbf{w}_{[k]}^T \mathbf{x}\right)$$

这种多分类的处理方式，我们称之为One-Versus-All(OVA) Decomposition。这种方法的优点是简单高效，可以使用logistic regression模型来解决；缺点是如果数据类别很多时，那么每次二分类问题中，正类和负类的数量差别就很大，数据不平衡unbalanced，这样会影响分类效果。但是，OVA还是非常常用的一种多分类算法。



One-Versus-All (OVA) Decomposition

1. for $k \in \mathcal{Y}$
   obtain $\mathbf{w}_{[k]}$ by running logistic regression on

$$\mathcal{D}_{[k]} = \{(\mathbf{x}_n, y'_n = 2 [\![y_n = k]\!] - 1)\}_{n=1}^{N}$$

2. return $g(\mathbf{x}) = \text{argmax}_{k \in \mathcal{Y}} \left(\mathbf{w}_{[k]}^T \mathbf{x} \right)$

- pros: efficient,
  can be coupled with any logistic regression-like approaches
- cons: often **unbalanced** $\mathcal{D}_{[k]}$ when K large
- extension: **multinomial ('coupled') logistic regression**

OVA: a simple multiclass **meta-algorithm**
to keep in your toolbox

四、Multiclass via Binary Classification

上一节，我们介绍了多分类算法OVA，但是这种方法存在一个问题，就是当类别k很多的时候，造成正负类数据unbalanced，会影响分类效果，表现不好。现在，我们介绍另一种方法来解决当k很大时，OVA带来的问题。

这种方法呢，每次只取两类进行binary classification，取值为{-1，+1}。假如k=4，那么总共需要进行$C_4^2=6$次binary classification。那么，六次分类之后，如果平面有个点，有三个分类器判断它是正方形，一个分类器判断是菱形，另外两个判断是三角形，那么取最多的那个，即判断它属于正方形，我们的分类就完成了。这种形式就如同k个足球对进行单循环的比赛，每场比赛都有一个队赢，一个队输，赢了得1分，输了得0分。那么总共进行了C_k^2次的比赛，最终取得分最高的那个队就可以了。



Multiclass Prediction: Combine **Pairwise** Classifiers

$$g(\mathbf{x}) = \text{tournament champion}\left\{\mathbf{w}_{[k,\ell]}^T\mathbf{x}\right\}$$

(voting of classifiers)

这种区别于OVA的多分类方法叫做One-Versus-One(OVO)。这种方法的优点是更加高效，因为虽然需要进行的分类次数增加了，但是每次只需要进行两个类别的比较，也就是说单次分类的数量减少了。而且一般不会出现数据unbalanced的情况。缺点是需要分类的次数多，时间复杂度和空间复杂度可能都比较高。



One-versus-one (OVO) Decomposition

1. for $(k,\ell) \in \mathcal{Y} \times \mathcal{Y}$
   obtain $\mathbf{w}_{[k,\ell]}$ by running linear binary classification on

$$\mathcal{D}_{[k,\ell]} = \{(\mathbf{x}_n, y_n' = 2\,[\![y_n = k]\!] - 1) : y_n = k \text{ or } y_n = \ell\}$$

2. return $g(\mathbf{x}) = $ tournament champion $\left\{\mathbf{w}_{[k,\ell]}^T \mathbf{x}\right\}$

- pros: efficient ('smaller' training problems), stable,
  can be coupled with any binary classification approaches
- cons: use $O(K^2)$ $\mathbf{w}_{[k,\ell]}$
  —more space, slower prediction, more training

OVO: another simple multiclass
meta-algorithm to keep in your toolbox

五、总结

本节课主要介绍了分类问题的三种线性模型：linear classification、linear regression和logistic regression。首先介绍了这三种linear models都可以来做binary classification。然后介绍了比梯度下降算法更加高效的SGD算法来进行logistic regression分析。最后讲解了两种多分类方法，一种是OVA，另一种是OVO。这两种方法各有优缺点，当类别数量k不多的时候，建议选择OVA，以减少分类次数。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程



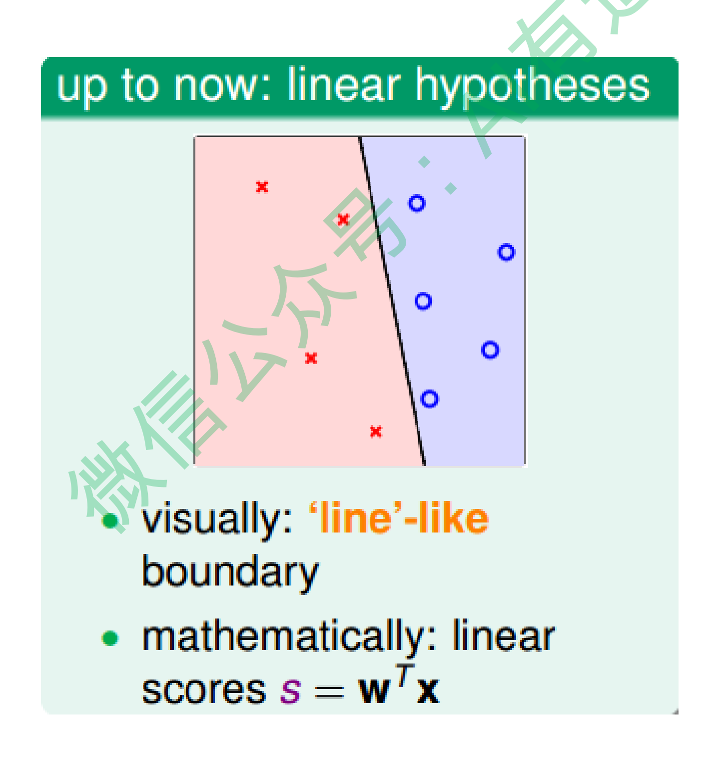
林轩田《机器学习基石》课程笔记12 -- Nonlinear Transformation

作者：红色石头　　公众号：AI有道（id：redstonewill）

上一节课，我们介绍了分类问题的三种线性模型，可以用来解决binary classification和multiclass classification问题。本节课主要介绍非线性的模型来解决分类问题。

一、Quadratic Hypothesis

之前介绍的线性模型，在2D平面上是一条直线，在3D空间中是一个平面。数学上，我们用线性得分函数s来表示：$s = w^T x$。其中，x为特征值向量，w为权重，s是线性的。



线性模型的优点就是，它的VC Dimension比较小，保证了$E_{in} \approx E_{out}$。但是缺点也很明显，对某些非线性问题，可能会造成E_{in}很大，虽然$E_{in} \approx E_{out}$，但是也造成E_{out}很大，分类效果不佳。



but limited . . .

- theoretically: d_{VC} **under control :-)**
- practically: on some $\mathcal{D}$, **large** E_{in} for every line **:-(**

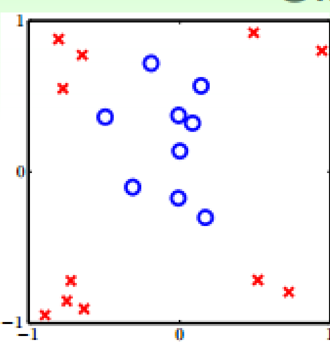
为了解决线性模型的缺点，我们可以使用非线性模型来进行分类。例如数据集D不是线性可分的，而是圆形可分的，圆形内部是正类，外面是负类。假设它的hypotheses可以写成：

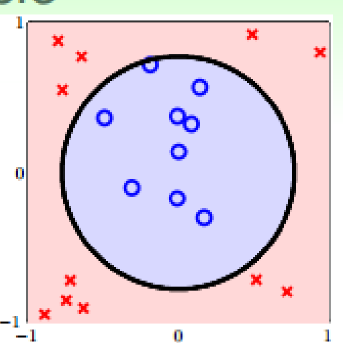
$$h_{SEP}(x) = sign(-x_1^2 - x_2^2 + 0.6)$$

基于这种非线性思想，我们之前讨论的PLA、Regression问题都可以有非线性的形式进行求解。



Circular Separable





- $\mathcal{D}$ not linear separable
- but **circular separable** by a circle of radius $\sqrt{0.6}$ centered at origin:

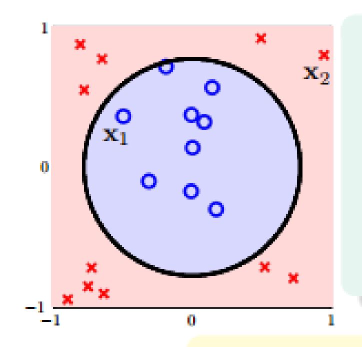
$$h_{\text{SEP}}(\mathbf{x}) = \text{sign}\left(-x_1^2 - x_2^2 + 0.6\right)$$

re-derive **Circular**-PLA, **Circular**-Regression, blahblah . . . all over again? :-)

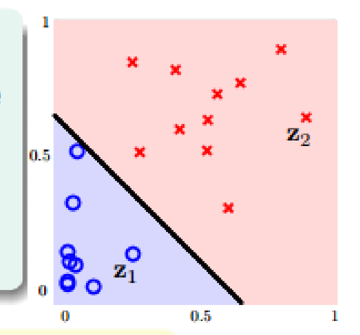
下面介绍如何设计这些非线性模型的演算法。还是上面介绍的平面圆形分类例子，它的h(x)的权重w0=0.6，w1=-1，w2=-1，但是h(x)的特征不是线性模型的$(1,x_1,x_2)$，而是$(1,x_1^2,x_2^2)$。我们令$z_0=1$，$z_1=x_1^2$，$z_2=x_2^2$，那么，h(x)变成：

$$h(x)=sign(\breve{w}_0\cdot z_0+\breve{w}_1\cdot z_1+\breve{w}_2\cdot z_2)=sign(0.6\cdot z_0-1\cdot z_1-1\cdot z_2)=sign(\breve{w}^Tz)$$

这种$x_n\rightarrow z_n$的转换可以看成是x空间的点映射到z空间中去，而在z域中，可以用一条直线进行分类，也就是从x空间的圆形可分映射到z空间的线性可分。z域中的直线对应于x域中的圆形。因此，我们把$x_n\rightarrow z_n$这个过程称之为特征转换（Feature Transform）。通过这种特征转换，可以将非线性模型转换为另一个域中的线性模型。



- $\{(\mathbf{x}_n, y_n)\}$ circular separable $\Longrightarrow \{(\mathbf{z}_n, y_n)\}$ linear separable
- $\mathbf{x} \in \mathcal{X} \overset{\Phi}{\longmapsto} \mathbf{z} \in \mathcal{Z}$: **(nonlinear) feature transform Φ**



circular separable in $\mathcal{X} \Longrightarrow$ linear separable in $\mathcal{Z}$

vice versa?



已知x域中圆形可分在z域中是线性可分的，那么反过来，如果在z域中线性可分，是否在x域中一定是圆形可分的呢？答案是否定的。由于权重向量w取值不同，x域中的hypothesis可能是圆形、椭圆、双曲线等等多种情况。

$$(z_0, z_1, z_2) = \mathbf{z} = \mathbf{\Phi}(\mathbf{x}) = (1, x_1^2, x_2^2)$$
$$h(\mathbf{x}) = \tilde{h}(\mathbf{z}) = \text{sign}\left(\tilde{\mathbf{w}}^T \mathbf{\Phi}(\mathbf{x})\right) = \text{sign}\left(\tilde{w}_0 + \tilde{w}_1 x_1^2 + \tilde{w}_2 x_2^2\right)$$

$\tilde{\mathbf{w}} = (\tilde{w}_0, \tilde{w}_1, \tilde{w}_2)$

- $(0.6, -1, -1)$: circle ($\circ$ inside)
- $(-0.6, +1, +1)$: circle ($\circ$ outside)
- $(0.6, -1, -2)$: ellipse
- $(0.6, -1, +2)$: hyperbola
- $(0.6, +1, +2)$: **constant $\circ$:-)**

目前讨论的x域中的圆形都是圆心过原点的，对于圆心不过原点的一般情况，$x_n \to z_n$映射公式包含的所有项为：

$$\Phi_2(x) = (1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$$

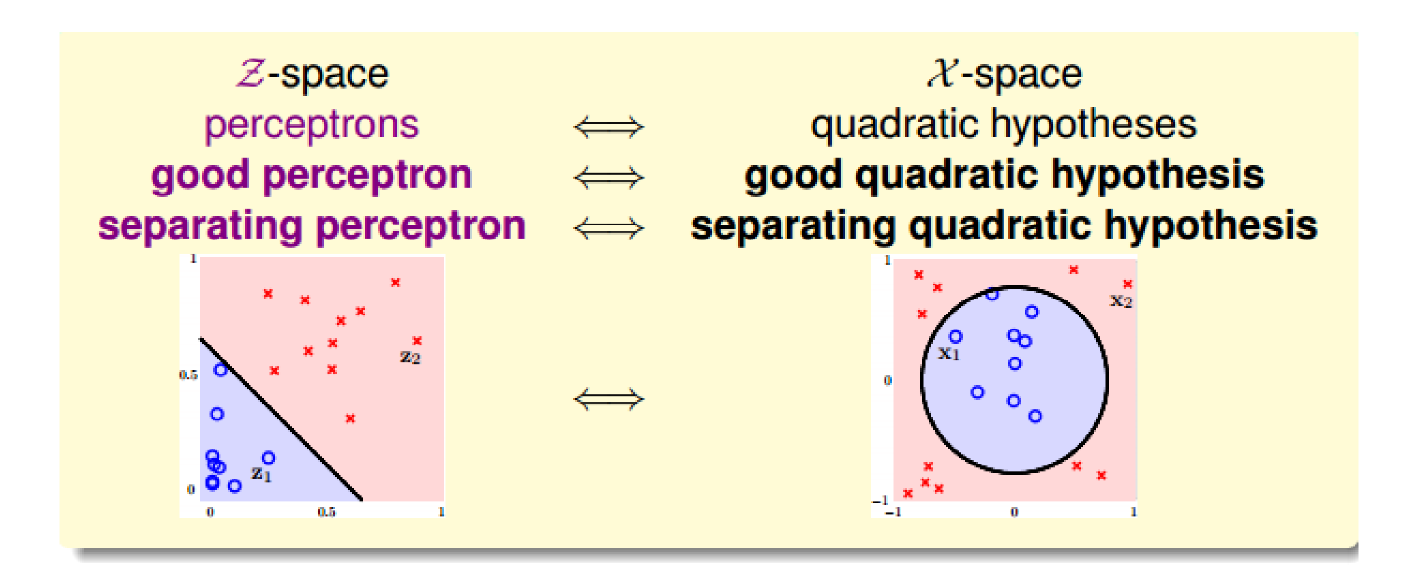
也就是说，对于二次hypothesis，它包含二次项、一次项和常数项1，z域中每一条线对应x域中的某二次曲线的分类方式，也许是圆，也许是椭圆，也许是双曲线等等。那么z域中的hypothesis可以写成：

$$\mathcal{H}_{\Phi_2} = \left\{ h(\mathbf{x}) : h(\mathbf{x}) = \tilde{h}(\mathbf{\Phi}_2(\mathbf{x})) \text{ for some linear } \tilde{h} \text{ on } \mathcal{Z} \right\}$$

二、Nonlinear Transform

上一部分我们定义了什么了二次hypothesis，那么这部分将介绍如何设计一个好的二次hypothesis来达到良好的分类效果。那么目标就是在z域中设计一个最佳的分类线。





- want: get **good perceptron** in $\mathcal{Z}$-space
- known: get **good perceptron** in $\mathcal{X}$-space with data $\{(\mathbf{x}_n, y_n)\}$

todo: get **good perceptron** in $\mathcal{Z}$-space with data $\{(\mathbf{z}_n = \mathbf{\Phi}_2(\mathbf{x}_n), y_n)\}$

其实，做法很简单，利用映射变换的思想，通过映射关系，把x域中的最高阶二次的多项式转换为z域中的一次向量，也就是从quardratic hypothesis转换成了perceptrons问题。用z值代替x多项式，其中向量z的个数与x域中x多项式的个数一致（包含常数项）。这样就可以在z域中利用线性分类模型进行分类训练。训练好的线性模型之后，再将z替换为x的多项式就可以了。具体过程如下：



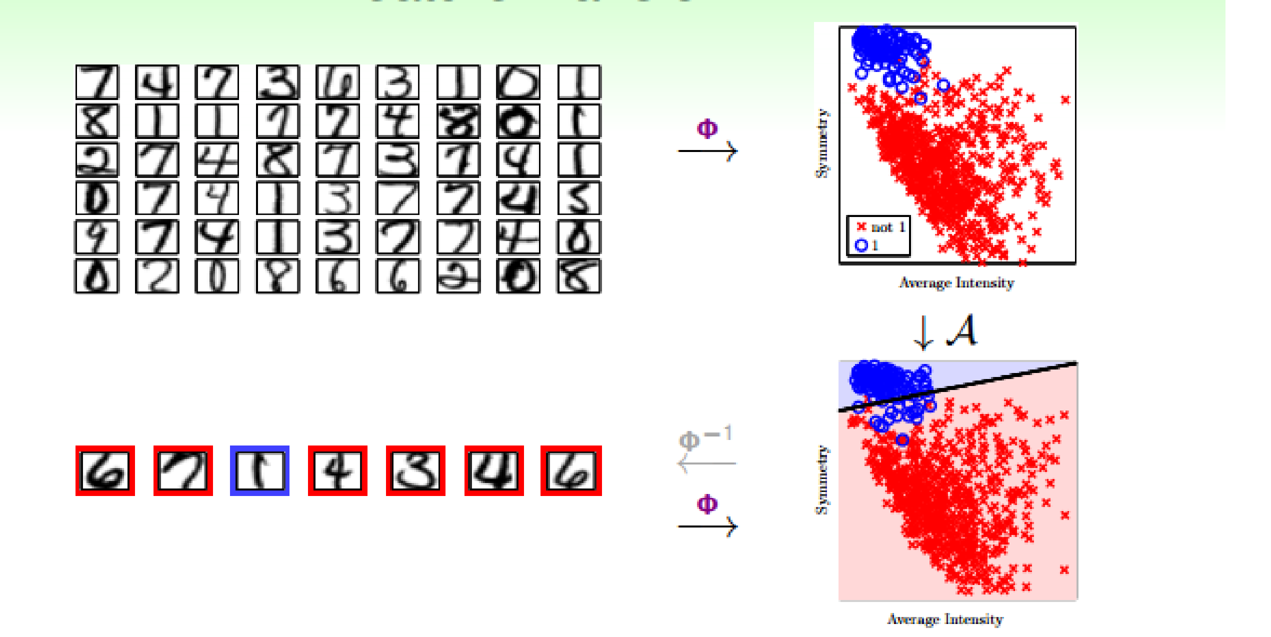
The Nonlinear Transform Steps

整个过程就是通过映射关系，换个空间去做线性分类，重点包括两个：

- 特征转换
- 训练线性模型

其实，我们以前处理机器学习问题的时候，已经做过类似的特征变换了。比如数字识别问题，我们从原始的像素值特征转换为一些实际的concrete特征，比如密度、对称性等等，这也用到了feature transform的思想。





not new, not just polynomial:

raw (pixels) $\xrightarrow{\text{domain knowledge}}$ concrete (intensity, symmetry)

三、Price of Nonlinear Transform

若x特征维度是d维的，也就是包含d个特征，那么二次多项式个数，即z域特征维度是：

$$\check{d} = 1 + C_d^1 + C_d^2 + d = \frac{d(d+3)}{2} + 1$$

如果x特征维度是2维的，即(x_1, x_2)，那么它的二次多项式为$(1, x_1, x_2, x_1^2, x_1 x_2, x_2^2)$，有6个。

现在，如果阶数更高，假设阶数为Q，那么对于x特征维度是d维的，它的z域特征维度为：

$$\check{d} = C_{Q+d}^{Q} = C_{Q+d}^{d} = O(Q^d)$$

由上式可以看出，计算z域特征维度个数的时间复杂度是Q的d次方，随着Q和d的增大，计算量会变得很大。同时，空间复杂度也大。也就是说，这种特征变换的一个代价是计算的时间、空间复杂度都比较大。



Q-th order polynomial transform: $\Phi_Q(\mathbf{x}) = \big(1, x_1, x_2, \ldots, x_d, x_1^2, x_1 x_2, \ldots, x_d^2, \ldots, x_1^Q, x_1^{Q-1}x_2, \ldots, x_d^Q \big)$

$\underbrace{1}_{\tilde{w}_0} + \underbrace{\tilde{d}}_{\text{others}}$ dimensions

= # ways of $\le Q$-combination from d kinds with repetitions

= $\binom{Q+d}{Q} = \binom{Q+d}{d} = O(Q^d)$

= efforts needed for computing/storing $\mathbf{z} = \Phi_Q(\mathbf{x})$ and $\tilde{\mathbf{w}}$

Q large $\Longrightarrow$ **difficult to compute/store**

另一方面，z域中特征个数随着Q和d增加变得很大，同时权重w也会增大，即自由度增加，VC Dimension增大。令z域中的特征维度是$1+\breve{d}$，则在在域中，任何$\breve{d}+2$的输入都不能被shattered；同样，在x域中，任何$\breve{d}+2$的输入也不能被shattered。$\breve{d}+1$是VC Dimension的上界，如果$\breve{d}+1$很大的时候，相应的VC Dimension就会很大。根据之前章节课程的讨论，VC Dimension过大，模型的泛化能力会比较差。

$\underbrace{1}_{\tilde{w}_0} + \underbrace{\tilde{d}}_{\text{others}}$ dimensions $= O(Q^d)$

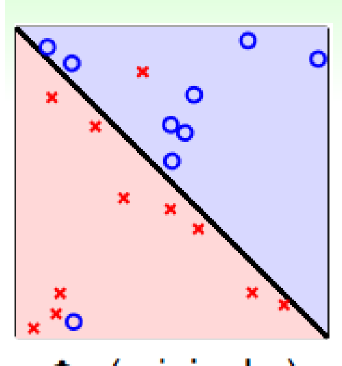
- number of free parameters $\tilde{w}_i = \tilde{d} + 1 \approx d_{\mathrm{VC}}(\mathcal{H}_{\Phi_Q})$
- $d_{\mathrm{VC}}(\mathcal{H}_{\Phi_Q}) \le \tilde{d} + 1$, why?

  any $\tilde{d}+2$ inputs not shattered in $\mathcal{Z}$
  $\Longrightarrow$ any $\tilde{d}+2$ inputs not shattered in $\mathcal{X}$

Q large $\Longrightarrow$ **large d_{VC}**

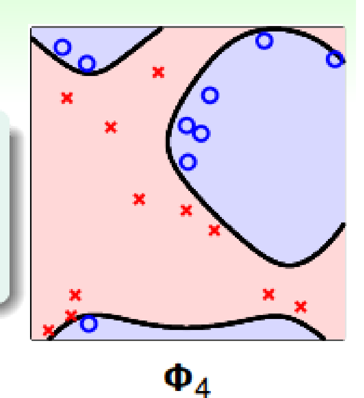
下面通过一个例子来解释为什么VC Dimension过大，会造成不好的分类效果：





which one do you prefer? :-)

- Φ_1 'visually' preferred
- Φ_4: $E_{in}(g) = 0$ but overkill



1. can we make sure that $E_{out}(g)$ is close enough to $E_{in}(g)$?
2. can we make $E_{in}(g)$ small enough?

trade-off:

$\tilde{d}$ (Q)	1	2
higher	:-(	**:-D**
lower	**:-D**	:-(

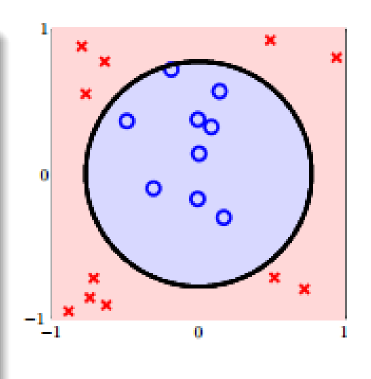
上图中，左边是用直线进行线性分类，有部分点分类错误；右边是用四次曲线进行非线性分类，所有点都分类正确，那么哪一个分类效果好呢？单从平面上这些训练数据来看，四次曲线的分类效果更好，但是四次曲线模型很容易带来过拟合的问题，虽然它的E_{in}比较小，从泛化能力上来说，还是左边的分类器更好一些。也就是说VC Dimension过大会带来过拟合问题，$\breve{d}+1$不能太大了。

那么如何选择合适的Q，来保证不会出现过拟合问题，使模型的泛化能力强呢？一般情况下，为了尽量减少特征自由度，我们会根据训练样本的分布情况，人为地减少、省略一些项。但是，这种人为地删减特征会带来一些"自我分析"代价，虽然对训练样本分类效果好，但是对训练样本外的样本，不一定效果好。所以，一般情况下，还是要保存所有的多项式特征，避免对训练样本的人为选择。

Visualize $\mathcal{X} = \mathbb{R}^2$

- full Φ_2: $\mathbf{z} = (1, x_1, x_2, x_1^2, x_1x_2, x_2^2)$, $d_{VC} = 6$
- or $\mathbf{z} = (1, x_1^2, x_2^2)$, $d_{VC} = 3$, **after visualizing**?
- or better $\mathbf{z} = (1, x_1^2 + x_2^2)$, $d_{VC} = 2$?
- or even better $\mathbf{z} = (\text{sign}(0.6 - x_1^2 - x_2^2))$?

—careful about **your brain's 'model complexity'**



for VC-safety, Φ shall be decided **without 'peeking'** data



四、Structured Hypothesis Sets

下面，我们讨论一下从x域到z域的多项式变换。首先，如果特征维度只有1维的话，那么变换多项式只有常数项：

$$\Phi_0(x) = (1)$$

如果特征维度是两维的，变换多项式包含了一维的$\Phi_0(x)$：

$$\Phi_1(x) = (\Phi_0(x), x_1, x_2, \ldots, x_d)$$

如果特征维度是三维的，变换多项式包含了二维的$\Phi_1(x)$：

$$\Phi_2(x) = (\Phi_1(x), x_1^2, x_1x_2, \ldots, x_d^2)$$

以此类推，如果特征维度是Q次，那么它的变换多项式为：

$$\Phi_Q(x) = (\Phi_{Q-1}(x), x_1^Q, x_1^{Q-1}x_2, \cdots, x_d^Q)$$

那么对于不同阶次构成的hypothesis有如下关系：

$$H_{\Phi_0} \subset H_{\Phi_1} \subset H_{\Phi_2} \subset \cdots \subset H_{\Phi_Q}$$

我们把这种结构叫做Structured Hypothesis Sets：

$$\begin{array}{ccccccccccc} \mathcal{H}_{\Phi_0} & \subset & \mathcal{H}_{\Phi_1} & \subset & \mathcal{H}_{\Phi_2} & \subset & \mathcal{H}_{\Phi_3} & \subset & \ldots & \subset & \mathcal{H}_{\Phi_Q} \\ \| & & \| & & \| & & \| & & & & \| \\ \mathcal{H}_0 & & \mathcal{H}_1 & & \mathcal{H}_2 & & \mathcal{H}_3 & & \ldots & & \mathcal{H}_Q \end{array}$$

$\mathcal{H}_0$ $\mathcal{H}_1$ $\mathcal{H}_2$ $\mathcal{H}_3$ $\cdots$

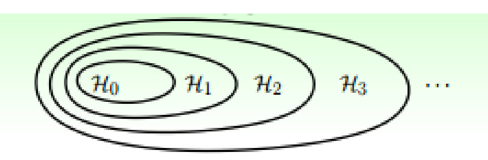
那么对于这种Structured Hypothesis Sets，它们的VC Dimension满足下列关系：

$$d_{VC}(H_0) \leq d_{VC}(H_1) \leq d_{VC}(H_2) \leq \cdots \leq d_{VC}(H_Q)$$

它的E_{in}满足下列关系：

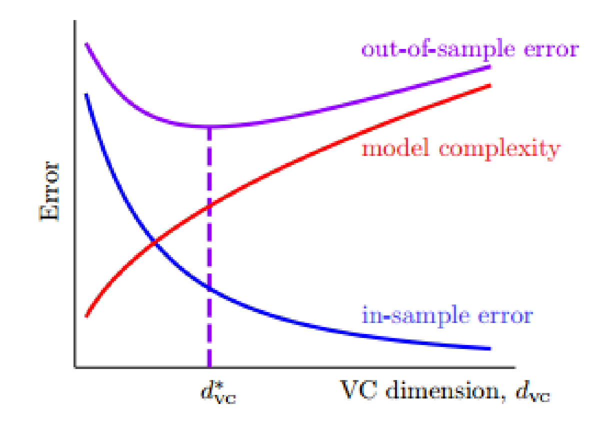
$$E_{in}(g_0) \geq E_{in}(g_1) \geq E_{in}(g_2) \geq \cdots \geq E_{in}(g_Q)$$





Let $g_i = \text{argmin}_{h \in \mathcal{H}_i} E_{\text{in}}(h)$:

$$
\begin{array}{ccccccccc}
\mathcal{H}_0 & \subset & \mathcal{H}_1 & \subset & \mathcal{H}_2 & \subset & \mathcal{H}_3 & \subset & \cdots \\
d_{\text{vc}}(\mathcal{H}_0) & \leq & d_{\text{vc}}(\mathcal{H}_1) & \leq & d_{\text{vc}}(\mathcal{H}_2) & \leq & d_{\text{vc}}(\mathcal{H}_3) & \leq & \cdots \\
E_{\text{in}}(g_0) & \geq & E_{\text{in}}(g_1) & \geq & E_{\text{in}}(g_2) & \geq & E_{\text{in}}(g_3) & \geq & \cdots
\end{array}
$$



use $\mathcal{H}_{1126}$ won't be good! :-(

从上图中也可以看到，随着变换多项式的阶数增大，虽然E_{in}逐渐减小，但是model complexity会逐渐增大，造成E_{out}很大，所以阶数不能太高。

那么，如果选择的阶数很大，确实能使E_{in}接近于0，但是泛化能力通常很差，我们把这种情况叫做tempting sin。所以，一般最合适的做法是先从低阶开始，如先选择一阶hypothesis，看看E_{in}是否很小，如果E_{in}足够小的话就选择一阶，如果E_{in}大的话，再逐渐增加阶数，直到满足要求为止。也就是说，尽量选择低阶的hypothes，这样才能得到较强的泛化能力。

- tempting sin: use $\mathcal{H}_{1126}$, low $E_{\text{in}}(g_{1126})$ to fool your boss —**really? :-(a dangerous path of no return**
- safe route: $\mathcal{H}_1$ first
  - if $E_{\text{in}}(g_1)$ good enough, **live happily thereafter :-)**
  - otherwise, move right of the curve **with nothing lost except 'wasted' computation**

linear model first:
simple, efficient, **safe**, **and workable!**

五、总结

这节课主要介绍了非线性分类模型，通过非线性变换，将非线性模型映射到另一个空间，转换为线性模型，再来进行线性分类。本节课完整介绍了非线性变换的整体流程，以及非线性变扌



可能会带来的一些问题：时间复杂度和空间复杂度的增加。最后介绍了在要付出代价的情况下，使用非线性变换的最安全的做法，尽可能使用简单的模型，而不是模型越复杂越好。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程

林轩田《机器学习基石》课程笔记13 -- Hazard of Overfitting

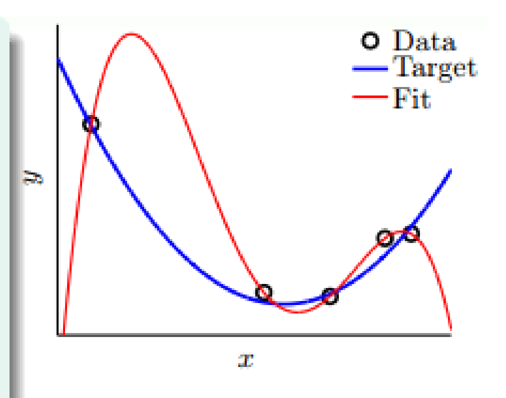
作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了非线性分类模型，通过非线性变换，将非线性模型映射到另一个空间，转换为线性模型，再来进行分类，分析了非线性变换可能会使计算复杂度增加。本节课介绍这种模型复杂度增加带来机器学习中一个很重要的问题：过拟合（overfitting）。

一、What is Overfitting?

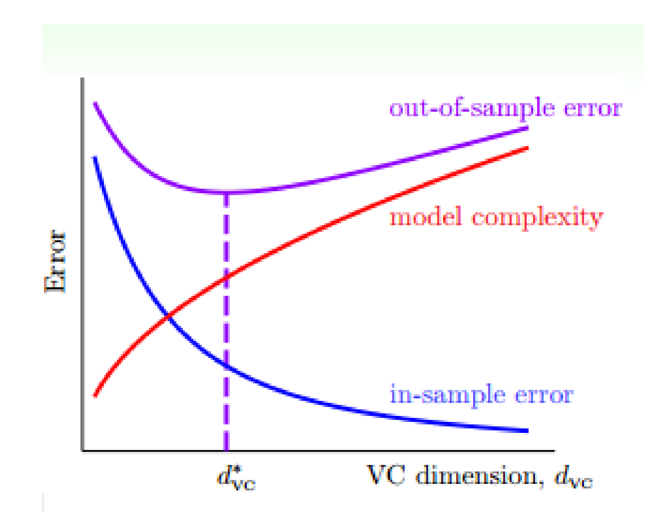
首先，我们通过一个例子来介绍什么bad generalization。假设平面上有5个点，目标函数f(x)是2阶多项式，如果hypothesis是二阶多项式加上一些小的noise的话，那么这5个点很靠近这个hypothesis，E_{in}很小。如果hypothesis是4阶多项式，那么这5点会完全落在hypothesis上，$E_{in}=0$。虽然4阶hypothesis的E_{in}比2阶hypothesis的要好很多，但是它的E_{out}很大。因为根据VC Bound理论，阶数越大，即VC Dimension越大，就会让模型复杂度更高，E_{out}更大。我们把这种E_{in}很小，E_{out}很大的情况称之为bad generation，即泛化能力差。

- regression for $x \in \mathbb{R}$ with $N = 5$ examples
- target $f(x)$ = 2nd order polynomial
- label $y_n = f(x_n) +$ very small noise
- linear regression in $\mathcal{Z}$-space + $\mathbf{\Phi}$ = 4th order polynomial
- unique solution passing all examples $\Longrightarrow E_{in}(g) = 0$
- $E_{out}(g)$ **huge**



bad generalization: low E_{in}, high E_{out}

我们回过头来看一下VC曲线：



hypothesis的阶数越高，表示VC Dimension越大。随着VC Dimension增大，E_{in}是一直减小的，而E_{out}先减小后增大。在d^*位置，E_{out}取得最小值。在d^*_{VC}右侧，随着VC Dimension越来越大，E_{in}越来越小，接近于0，E_{out}越来越大。即当VC Dimension很大的时候，这种对训练样本拟合过分好的情况称之为过拟合（overfitting）。另一方面，在d^*_{VC}左侧，随着VC Dimension越来越小，E_{in}和E_{out}都越来越大，这种情况称之为欠拟合（underfitting），即模型对训练样本的拟合度太差，VC Dimension太小了。

- take $d_{VC} = 1126$ for learning:
  bad generalization
  —(E_{out} - E_{in}) large
- switch from $d_{VC} = d^*_{VC}$ to $d_{VC} = 1126$:
  overfitting
  —$E_{in} \downarrow$, $E_{out} \uparrow$
- switch from $d_{VC} = d^*_{VC}$ to $d_{VC} = 1$:
  underfitting
  —$E_{in} \uparrow$, $E_{out} \uparrow$

bad generation和overfitting的关系可以理解为：overfitting是VC Dimension过大的一个过程，bad generation是overfitting的结果。

bad generalization: low E_{in}, high E_{out};
overfitting: lower E_{in}, higher E_{out}

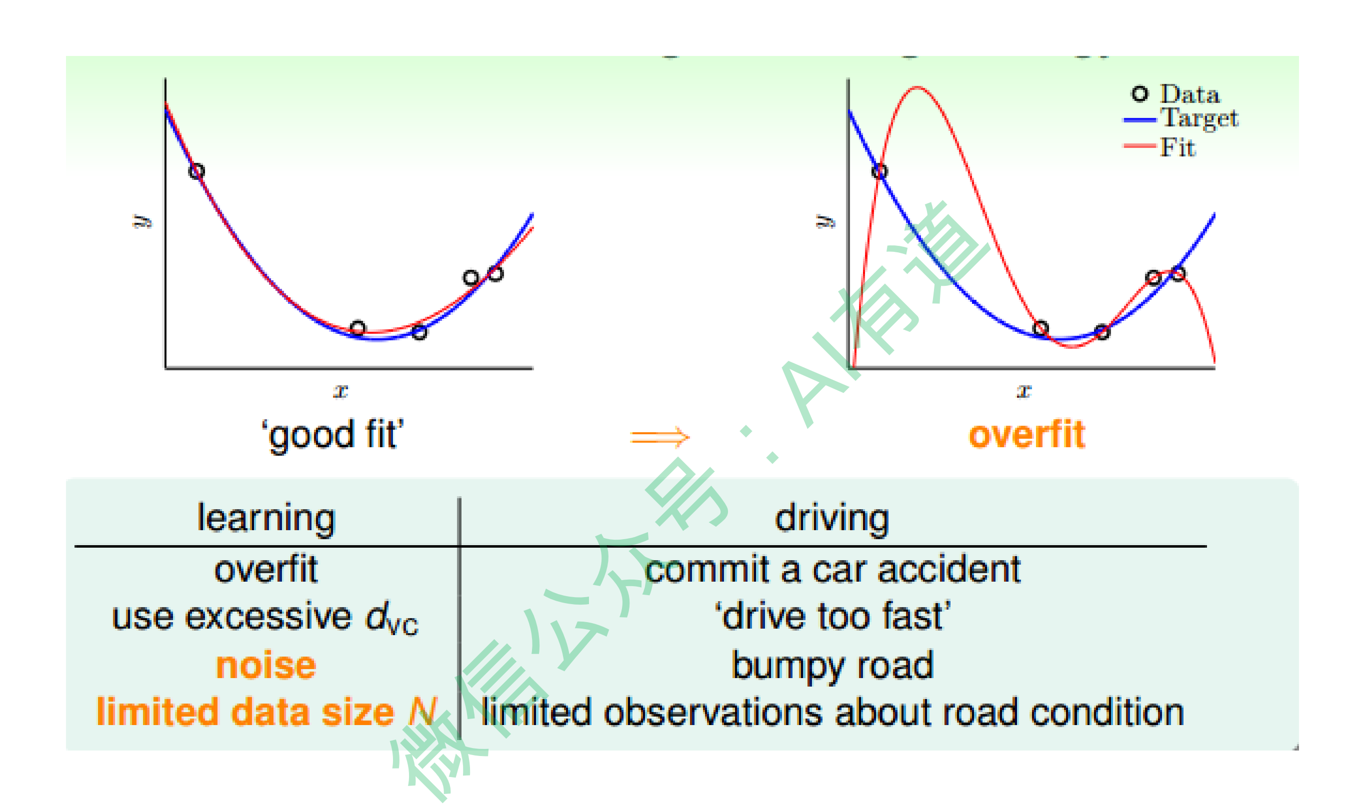


一个好的fit，E_{in}和E_{out}都比较小，尽管E_{in}没有足够接近零；而对overfitting来说，$E_{in} \approx 0$，但是E_{out}很大。那么，overfitting的原因有哪些呢？

我们举个开车的例子，把发生车祸比作成overfitting，那么造成车祸的原因包括：

- **车速太快（VC Dimension太大）；**
- **道路崎岖（noise）；**
- **对路况的了解程度（训练样本数量N不够）；**

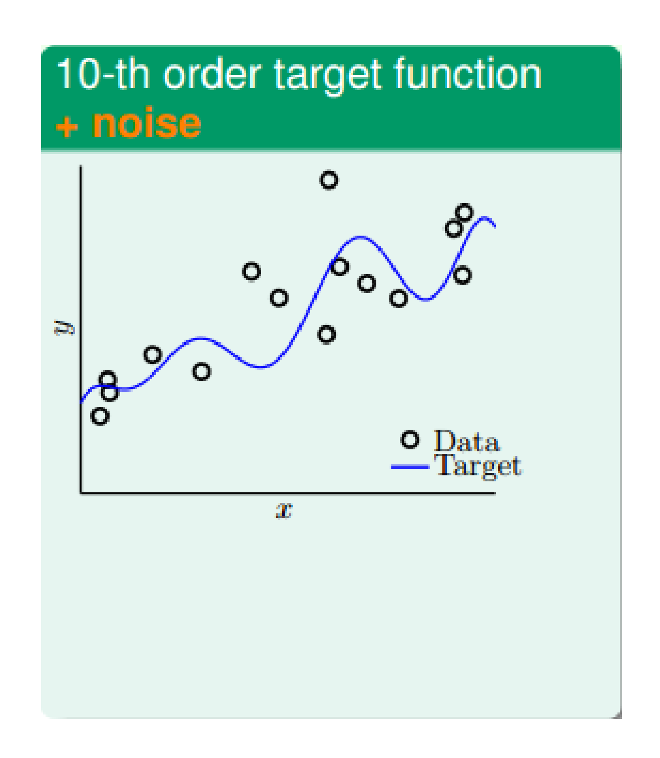
也就是说，VC Dimension、noise、N这三个因素是影响过拟合现象的关键。



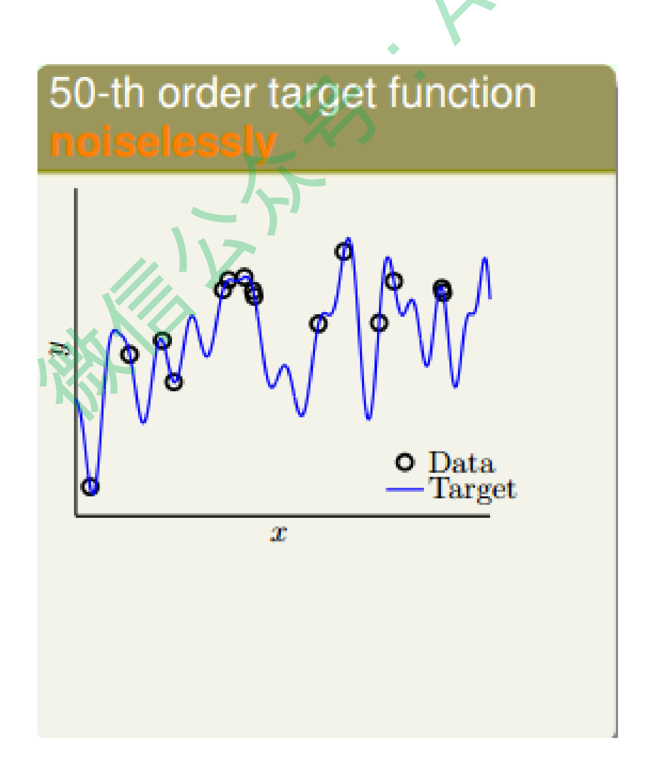
二、The Role of Noise and Data Size

为了尽可能详细地解释overfitting，我们进行这样一个实验，试验中的数据集不是很大。首先，在二维平面上，一个模型的分布由目标函数f(x)（x的10阶多项式）加上一些noise构成，下图中，离散的圆圈是数据集，目标函数是蓝色的曲线。数据没有完全落在曲线上，是因为加入了noise。



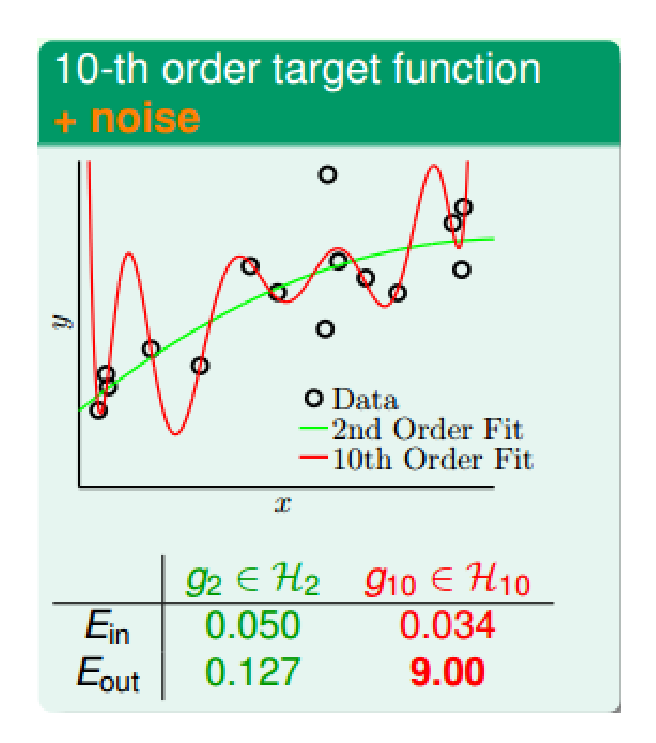


然后，同样在二维平面上，另一个模型的分布由目标函数f(x)（x的50阶多项式）构成，没有加入noise。下图中，离散的圆圈是数据集，目标函数是蓝色的曲线。可以看出由于没有noise，数据集完全落在曲线上。



现在，有两个学习模型，一个是2阶多项式，另一个是10阶多项式，分别对上面两个问题进行建模。首先，对于第一个目标函数是10阶多项式包含noise的问题，这两个学习模型的效果如下图所示：

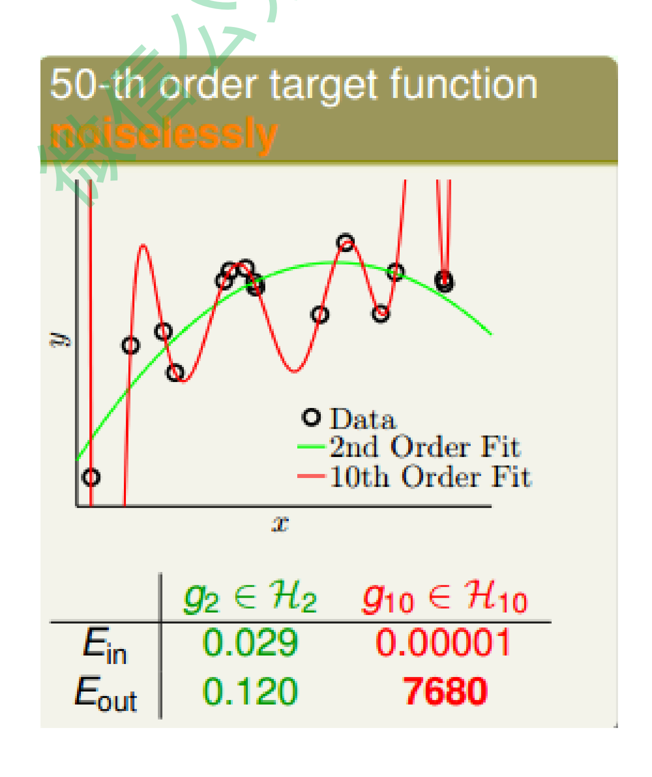




	$g_2 \in \mathcal{H}_2$	$g_{10} \in \mathcal{H}_{10}$
E_{in}	0.050	0.034
E_{out}	0.127	**9.00**

由上图可知，2阶多项式的学习模型$E_{in}=0.050$，$E_{out}=0.127$；10阶多项式的学习模型$E_{in}=0.034$，$E_{out}=9.00$。虽然10阶模型的E_{in}比2阶的小，但是其E_{out}要比2阶的大得多，而2阶的E_{in}和E_{out}相差不大，很明显用10阶的模型发生了过拟合。

然后，对于第二个目标函数是50阶多项式没有noise的问题，这两个学习模型的效果如下图所示：



	$g_2 \in \mathcal{H}_2$	$g_{10} \in \mathcal{H}_{10}$
E_{in}	0.029	0.00001
E_{out}	0.120	**7680**

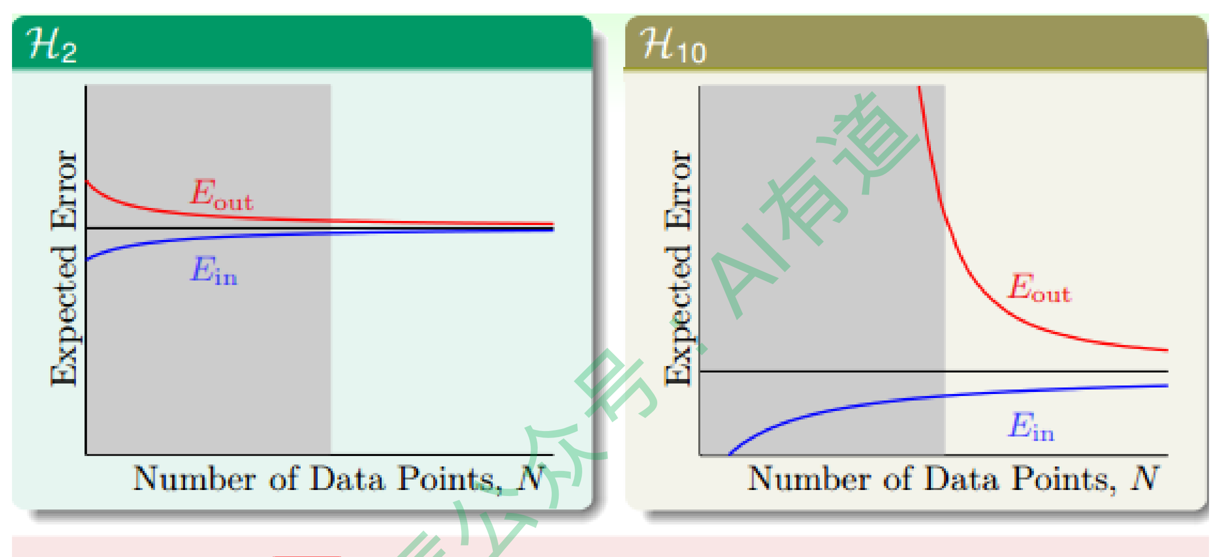
由上图可知，2阶多项式的学习模型$E_{in}=0.029$，$E_{out}=0.120$；10阶多项式的学习模型$E_{in}=0.00001$，$E_{out}=7680$。虽然10阶模型的E_{in}比2阶的小，但是其E_{out}要比2阶的大得多的多，而2阶的E_{in}和E_{out}相差不大，很明显用10阶的模型仍



发生了明显的过拟合。

上面两个问题中，10阶模型都发生了过拟合，反而2阶的模型却表现得相对不错。这好像违背了我们的第一感觉，比如对于目标函数是10阶多项式，加上noise的模型，按道理来说应该是10阶的模型更能接近于目标函数，因为它们阶数相同。但是，事实却是2阶模型泛化能力更强。这种现象产生的原因，从哲学上来说，就是“以退为进”。有时候，简单的学习模型反而能表现的更好。

下面从learning curve来分析一下具体的原因，learning curve描述的是E_{in}和E_{out}随着数据量N的变化趋势。下图中左边是2阶学习模型的learning curve，右边是10阶学习模型的learning curve。



我们的第9次课的笔记 NTU林轩田机器学习基石课程学习笔记9 -- Linear Regression 已经介绍过了learning curve。在learning curve中，横轴是样本数量N，纵轴是Error。E_{in}和E_{out}可表示为：

$$E_{in} = noiselevel * (1 - \frac{d+1}{N})$$

$$E_{out} = noiselevel * (1 + \frac{d+1}{N})$$

其中d为模型阶次，左图中d=2，右图中d=10。

本节的实验问题中，数据量N不大，即对应于上图中的灰色区域。左图的灰色区域中，因为d=2，E_{in}和E_{out}相对来说比较接近；右图中的灰色区域中，d=10，根据



E_{in}和E_{out}的表达式，E_{in}很小，而E_{out}很大。这就解释了之前2阶多项式模型的E_{in}更接近E_{out}，泛化能力更好。

值得一提的是，如果数据量N很大的时候，上面两图中E_{in}和E_{out}都比较接近，但是对于高阶模型，z域中的特征很多的时候，需要的样本数量N很大，且容易发生维度灾难。关于维度灾难的详细生动解释，请参考我另一篇博文：

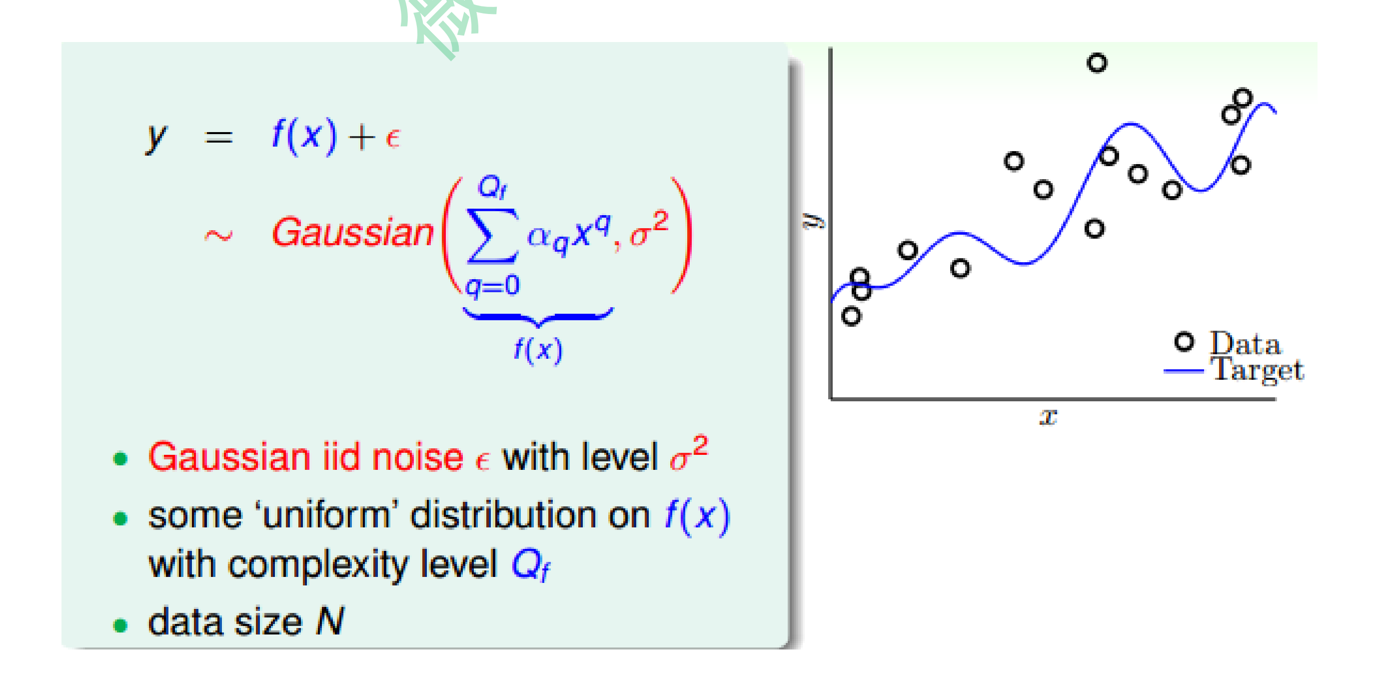
机器学习中的维度灾难

另一个例子中，目标函数是50阶多项式，且没有加入noise。这种情况下，我们发现仍然是2阶的模型拟合的效果更好一些，明明没有noise，为什么是这样的结果呢?

实际上，我们忽略了一个问题：这种情况真的没有noise吗？其实，当模型很复杂的时候，即50阶多项式的目标函数，无论是2阶模型还是10阶模型，都不能学习的很好，这种复杂度本身就会引入一种‘noise’。所以，这种高阶无noise的问题，也可以类似于10阶多项式的目标函数加上noise的情况，只是二者的noise有些许不同，下面一部分将会详细解释。

三、Deterministic Noise

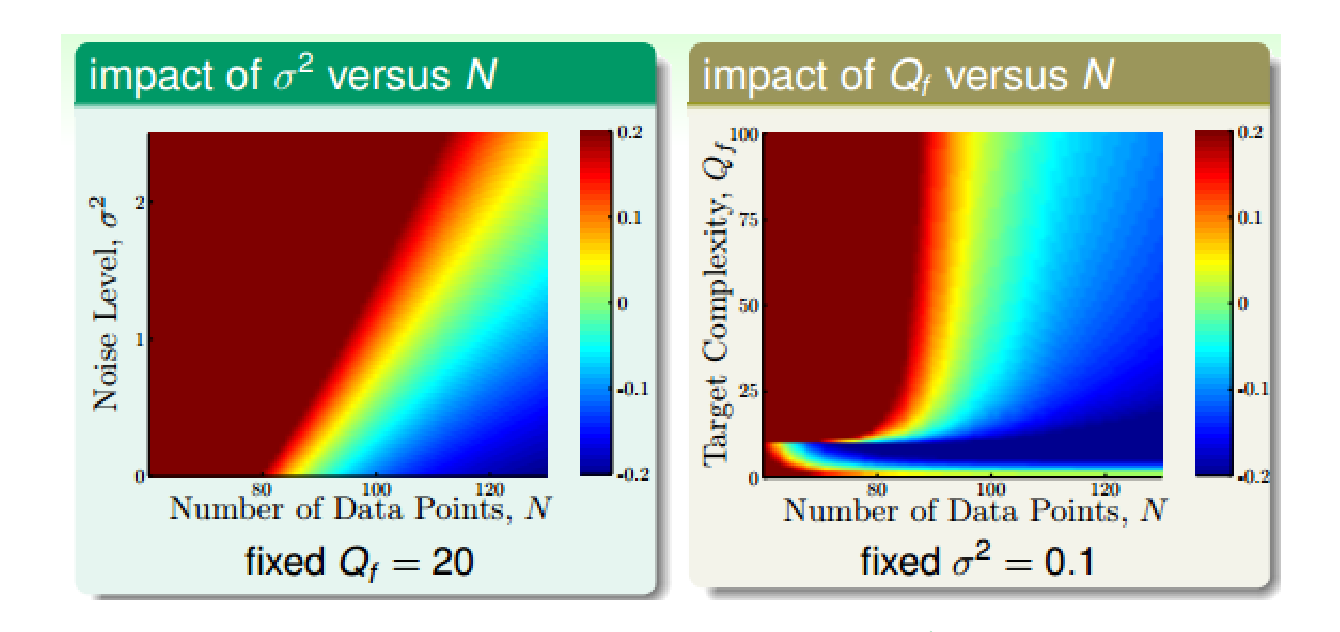
下面我们介绍一个更细节的实验来说明 什么时候小心overfit会发生。假设我们产生的数据分布由两部分组成：第一部分是目标函数f(x)，Q_f阶多项式；第二部分是噪声ϵ，服从Gaussian分布。接下来我们分析的是noise强度不同对overfitting有什么样的影响。总共的数据量是N。



那么下面我们分析不同的(N, σ^2)和(N, Q_f)对overfit的影响。overfit可以量化为



$E_{out} - E_{in}$。结果如下：



上图中，红色越深，代表overfit程度越高，蓝色越深，代表overfit程度越低。先看左边的图，左图中阶数Q_f固定为20，横坐标代表样本数量N，纵坐标代表噪声水平σ^2。红色区域集中在N很小或者σ^2很大的时候，也就是说N越大，σ^2越小，越不容易发生overfit。右边图中$\sigma^2 = 0.1$，横坐标代表样本数量N，纵坐标代表目标函数阶数Q_f。红色区域集中在N很小或者Q_f很大的时候，也就是说N越大，Q_f越小，越不容易发生overfit。上面两图基本相似。

从上面的分析，我们发现σ^2对overfit是有很大的影响的，我们把这种noise称之为stochastic noise。同样地，Q_f即模型复杂度也对overfit有很大影响，而且二者影响是相似的，所以我们把这种称之为deterministic noise。之所以把它称为noise，是因为模型高复杂度带来的影响。

总结一下，有四个因素会导致发生overfitting：

- **data size N** $\downarrow$
- **stochastic noise** σ^2 $\uparrow$
- **deterministic noise** Q_f $\uparrow$
- **excessive power** $\uparrow$

我们刚才解释了如果目标函数f(x)的复杂度很高的时候，那么跟有noise也没有什么两样。因为目标函数很复杂，那么再好的hypothesis都会跟它有一些差距，我们把这种差距称之为deterministic noise。deterministic noise与stochastic noise不同，但是效果一样。其实deterministic noise类似于一个伪随机数发生器，它不会产生真正的随机数，而只产生伪随机数。它的值与hypothesis有关，且固定点x的deterministic noise



是固定的。

四、Dealing with Overfitting

现在我们知道了什么是overfitting，和overfitting产生的原因，那么如何避免overfitting呢？避免overfitting的方法主要包括：

- **start from simple model**
- **data cleaning/pruning**
- **data hinting**
- **regularization**
- **validataion**

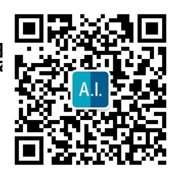
这几种方法类比于之前举的开车的例子，对应如下：

learning	driving
overfit	commit a car accident
use excessive d_{VC}	'drive too fast'
noise	bumpy road
limited data size N	limited observations about road condition
start from simple model	drive slowly
data cleaning/pruning	use more accurate road information
data hinting	exploit more road information
regularization	put the brakes
validation	monitor the dashboard

regularization和validation我们之后的课程再介绍，本节课主要介绍简单的data cleaning/pruning和data hinting两种方法。

data cleaning/pruning就是对训练数据集里label明显错误的样本进行修正（data cleaning），或者对错误的样本看成是noise，进行剔除（data pruning）。data cleaning/pruning关键在于如何准确寻找label错误的点或者是noise的点，而且如果这些点相比训练样本N很小的话，这种处理效果不太明显。

data hinting是针对N不够大的情况，如果没有办法获得更多的训练集，那么data hinting就可以对已知的样本进行简单的处理、变换，从而获得更多的样本。举个例子，数字分类问题，可以对已知的数字图片进行轻微的平移或者旋转，从而让N丰富起来，达到扩大训练集的目的。这种额外获得的例子称之为virtual examples。但是要



注意一点的就是，新获取的virtual examples可能不再是iid某个distribution。所以新构建的virtual examples要尽量合理，且是独立同分布的。

五、总结

本节课主要介绍了overfitting的概念，即当E_{in}很小，E_{out}很大的时候，会出现overfitting。详细介绍了overfitting发生的四个常见原因data size N、stochastic noise、deterministic noise和excessive power。解决overfitting的方法有很多，本节课主要介绍了data cleaning/pruning和data hinting两种简单的方法，之后的课程将会详细介绍regularization和validataion两种更重要的方法。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程





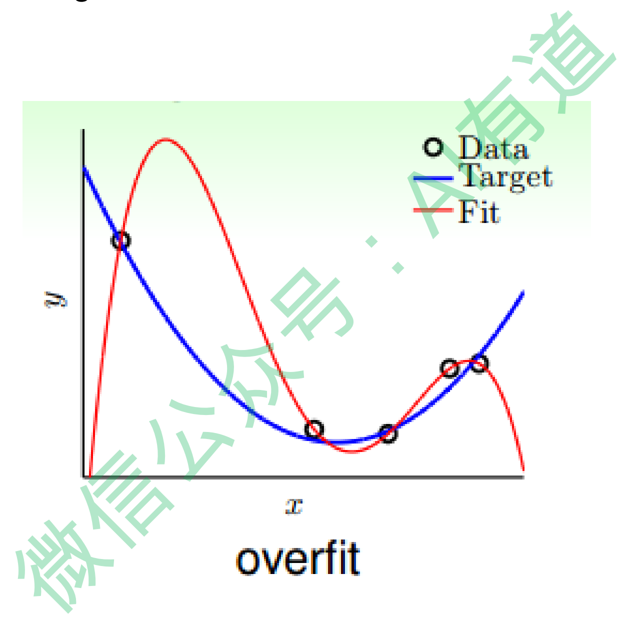
林轩田《机器学习基石》课程笔记14 -- Regularization

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们介绍了过拟合发生的原因：excessive power, stochastic/deterministic noise 和limited data。并介绍了解决overfitting的简单方法。本节课，我们将介绍解决overfitting的另一种非常重要的方法：Regularization规则化。

一、Regularized Hypothesis Set

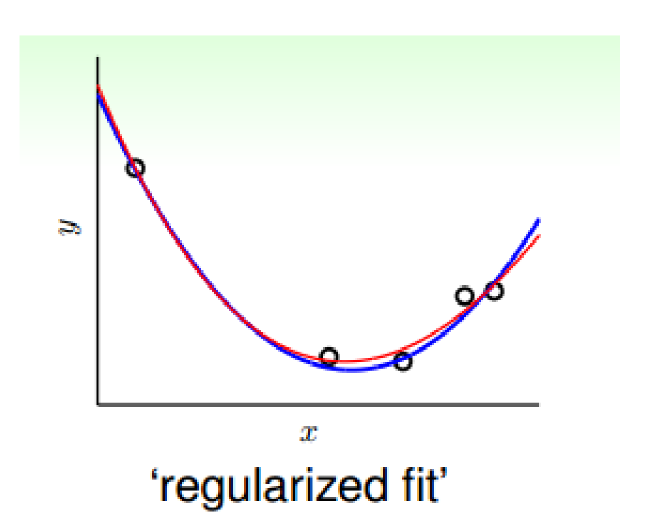
先来看一个典型的overfitting的例子：



如图所示，在数据量不够大的情况下，如果我们使用一个高阶多项式（图中红色曲线所示），例如10阶，对目标函数（蓝色曲线）进行拟合。拟合曲线波动很大，虽然E_{in}很小，但是E_{out}很大，也就造成了过拟合现象。

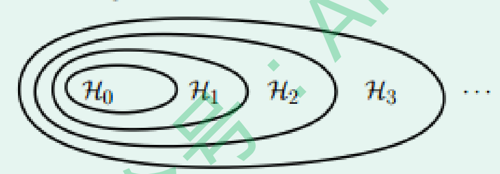
那么如何对过拟合现象进行修正，使hypothesis更接近于target function呢？一种方法就是regularized fit。





这种方法得到的红色fit曲线，要比overfit的红色曲线平滑很多，更接近与目标函数，它的阶数要更低一些。那么问题就变成了我们要把高阶（10阶）的hypothesis sets转换为低阶（2阶）的hypothesis sets。通过下图我们发现，不同阶数的hypothesis存在如下包含关系：

- idea: 'step back' from $\mathcal{H}_{10}$ to $\mathcal{H}_2$



- name history: function approximation for **ill-posed problems**

我们发现10阶多项式hypothesis sets里包含了2阶多项式hypothesis sets的所有项，那么在H_{10}中加入一些限定条件，使它近似为H_2即可。这种函数近似曾被称之为不适定问题（ill-posed problem）。

如何从10阶转换为2阶呢？首先，H_{10}可表示为：

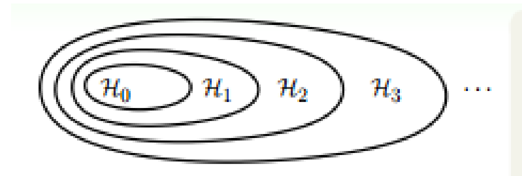
$$H_{10} = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \cdots + w_{10} x^{10}$$

而H_2可表示为：

$$H_2 = w_0 + w_1 x + w_2 x^2$$

所以，如果限定条件是$w_3 = w_4 = \cdots = w_{10} = 0$，那么就有$H_2 = H_{10}$。也就是说，对于高阶的hypothesis，为了防止过拟合，我们可以将其高阶部分的权重w限制为0，这样，就相当于从高阶的形式转换为低阶，fit波形更加平滑，不容易发生过拟合。





Q-th order polynomial transform for $x \in \mathbb{R}$:

$$\Phi_Q(x) = (1, x, x^2, \ldots, x^Q)$$

+ linear regression, denote $\tilde{\mathbf{w}}$ by $\mathbf{w}$

hypothesis $\mathbf{w}$ in $\mathcal{H}_{10}$: $w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_{10} x^{10}$

hypothesis $\mathbf{w}$ in $\mathcal{H}_2$: $w_0 + w_1 x + w_2 x^2$

that is, $\mathcal{H}_2 = \mathcal{H}_{10}$ AND 'constraint that $w_3 = w_4 = \ldots = w_{10} = 0$'

step back = **constraint**

那有一个问题，令H_{10}高阶权重w为0，为什么不直接使用H_2呢？这样做的目的是拓展我们的视野，为即将讨论的问题做准备。刚刚我们讨论的限制是H_{10}高阶部分的权重w限制为0，这是比较苛刻的一种限制。下面，我们把这个限制条件变得更宽松一点，即令任意8个权重w为0，并不非要限定$w_3 = w_4 = \cdots = w_{10} = 0$，这个Looser Constraint可以写成：

$$\sum_{q=0}^{10}(w_q \neq 0) \leq 3$$

也就只是限定了w不为0的个数，并不限定必须是高阶的w。这种hypothesis记为H_2'，称为sparse hypothesis set，它与H_2和H_{10}的关系为：

$$H_2 \subset H_2' \subset H_{10}$$

- more flexible than $\mathcal{H}_2$: $\mathcal{H}_2 \subset \mathcal{H}_2'$
- less risky than $\mathcal{H}_{10}$: $\mathcal{H}_2' \subset \mathcal{H}_{10}$

Looser Constraint对应的hypothesis应该更好解一些，但事实是sparse hypothesis set H_2'被证明也是NP-hard，求解非常困难。所以，还要转换为另一种易于求解的限定条件。

那么，我们寻找一种更容易求解的宽松的限定条件Softer Constraint，即：

$$\sum_{q=0}^{10} w_q^2 = ||w||^2 \leq C$$



其中，C是常数，也就是说，所有的权重w的平方和的大小不超过C，我们把这种hypothesis sets记为$H(C)$。

H_2'与$H(C)$的关系是，它们之间有重叠，有交集的部分，但是没有完全包含的关系，也不一定相等。对应$H(C)$，C值越大，限定的范围越大，即越宽松：

$$H(0) \subset H(1.126) \subset \cdots \subset H(1126) \subset \cdots \subset H(\infty) = H_{10}$$

当C无限大的时候，即限定条件非常宽松，相当于没有加上任何限制，就与H_{10}没有什么两样。$H(C)$称为regularized hypothesis set，这种形式的限定条件是可以进行求解的，我们把求解的满足限定条件的权重w记为w_{REG}。接下来就要探讨如何求解w_{REG}。

二、Weight Decay Regularization

现在，针对H(c)，即加上限定条件，我们的问题变成：

$$\min_{\mathbf{w}\in\mathbb{R}^{Q+1}} \quad E_{in}(\mathbf{w}) = \frac{1}{N}\underbrace{\sum_{n=1}^{N}(\mathbf{w}^T\mathbf{z}_n - y_n)^2}_{(\mathbf{Z}\mathbf{w}-\mathbf{y})^T(\mathbf{Z}\mathbf{w}-\mathbf{y})}$$

$$\text{s.t.} \quad \underbrace{\sum_{q=0}^{Q} w_q^2}_{\mathbf{w}^T\mathbf{w}} \le C$$

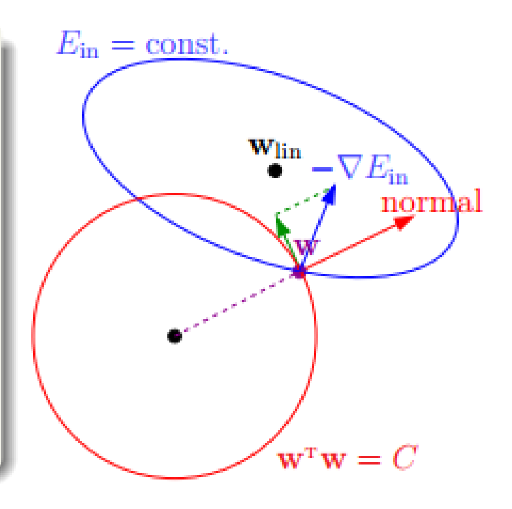
我们的目的是计算$E_{in}(w)$的最小值，限定条件是$||w^2|| \le C$。这个限定条件从几何角度上的意思是，权重w被限定在半径为$\sqrt{C}$的圆内，而球外的w都不符合要求，即便它是靠近$E_{in}(w)$梯度为零的w。

$$\min_{\mathbf{w}\in\mathbb{R}^{Q+1}} \quad E_{in}(\mathbf{w}) = \frac{1}{N}(\mathbf{Z}\mathbf{w}-\mathbf{y})^T(\mathbf{Z}\mathbf{w}-\mathbf{y}) \text{ s.t. } \mathbf{w}^T\mathbf{w} \le C$$

下面用一张图来解释在限定条件下，最小化$E_{in}(w)$的过程：



- decreasing direction: $-\nabla E_{in}(\mathbf{w})$, **remember? :-)**
- normal vector of $\mathbf{w}^T\mathbf{w} = C$: $\mathbf{w}$
- if $-\nabla E_{in}(\mathbf{w})$ and $\mathbf{w}$ not parallel: can **decrease $E_{in}(\mathbf{w})$ without violating the constraint**
- at optimal solution $\mathbf{w}_{REG}$, $-\nabla E_{in}(\mathbf{w}_{REG}) \propto \mathbf{w}_{REG}$



如上图所示，假设在空间中的一点w，根据梯度下降算法，w会朝着$-\nabla E_{in}$的方向移动（图中蓝色箭头指示的方向），在没有限定条件的情况下，w最终会取得最小值w_{lin}，即“谷底”的位置。现在，加上限定条件，即w被限定在半径为$\sqrt{C}$的圆内，w距离原点的距离不能超过圆的半径，球如图中红色圆圈所示$w^T w = C$。那么，这种情况下，w不能到达w_{lin}的位置，最大只能位于圆上，沿着圆的切线方向移动（图中绿色箭头指示的方向）。与绿色向量垂直的向量（图中红色箭头指示的方向）是圆切线的法向量，即w的方向，w不能靠近红色箭头方向移动。那么随着迭代优化过程，只要$-\nabla E_{in}$与w点切线方向不垂直，那么根据向量知识，$-\nabla E_{in}$一定在w点切线方向上有不为零的分量，即w点会继续移动。只有当$-\nabla E_{in}$与绿色切线垂直，即与红色法向量平行的时候，$-\nabla E_{in}$在切线方向上没有不为零的分量了，也就表示这时w达到了最优解的位置。

有了这个平行的概念，我们就得到了获得最优解需要满足的性质：

$$\nabla E_{in}(w_{REG}) + \frac{2\lambda}{N} w_{REG} = 0$$

上面公式中的λ称为Lagrange multiplier，是用来解有条件的最佳化问题常用的数学工具，$\frac{2}{N}$是方便后面公式推导。那么我们的目标就变成了求解满足上面公式的w_{REG}。

之前我们推导过，线性回归的E_{in}的表达式为：

$$E_{in} = \frac{1}{N}\sum_{n=1}^{N}(x_n^T w - y_n)^2$$

计算E_{in}梯度，并代入到平行条件中，得到：

$$\frac{2}{N}(Z^T Z w_{REG} - Z^T y) + \frac{2\lambda}{N} w_{REG} = 0$$

这是一个线性方程式，直接得到w_{REG}为：



$$w_{REG} = (Z^TZ + \lambda I)^{-1}Z^Ty$$

上式中包含了求逆矩阵的过程，因为Z^TZ是半正定矩阵，如果λ大于零，那么$Z^TZ + \lambda I$一定是正定矩阵，即一定可逆。另外提一下，统计学上把这叫做ridge regression，可以看成是linear regression的进阶版。

如果对于更一般的情况，例如逻辑回归问题中，∇E_{in}不是线性的，那么将其代入平行条件中得到的就不是一个线性方程式，w_{REG}不易求解。下面我们从另一个角度来看一下平行等式：

$$\nabla E_{in}(w_{REG}) + \frac{2\lambda}{N}w_{REG} = 0$$

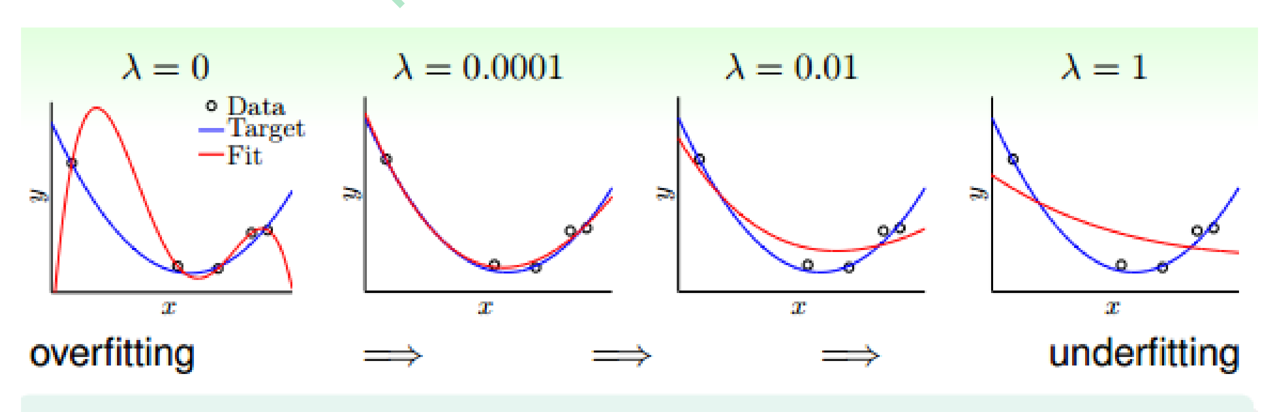
已知∇E_{in}是E_{in}对w_{REG}的导数，而$\frac{2\lambda}{N}w_{REG}$也可以看成是$\frac{\lambda}{N}w_{REG}^2$的导数。那么平行等式左边可以看成一个函数的导数，导数为零，即求该函数的最小值。也就是说，问题转换为最小化该函数：

$$E_{aug}(w) = E_{in}(w) + \frac{\lambda}{N}w^Tw$$

该函数中第二项就是限定条件regularizer，也称为weight-decay regularization。我们把这个函数称为Augmented Error，即$E_{aug}(w)$。

如果λ不为零，对应于加上了限定条件，若λ等于零，则对应于没有任何限定条件，问题转换成之前的最小化$E_{in}(w)$。

下面给出一个曲线拟合的例子，λ取不同的值时，得到的曲线也不相同：



从图中可以看出，当$\lambda = 0$时，发生了过拟合；当$\lambda = 0.0001$时，拟合的效果很好；当$\lambda = 0.01$和$\lambda = 1$时，发生了欠拟合。我们可以把λ看成是一种penality，即对hypothesis复杂度的惩罚，λ越大，w就越小，对应于C值越小，即这种惩罚越大，拟合曲线就会越平滑，高阶项就会削弱，容易发生欠拟合。λ一般取比较小的值就能达

到良好的拟合效果，过大过小都有问题，但究竟取什么值，要根据具体训练数据和模型进行分析与调试。

call '$+\frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$' **weight-decay** regularization:

larger λ
$\Longleftrightarrow$ prefer shorter $\mathbf{w}$
$\Longleftrightarrow$ effectively smaller C

—go with 'any' transform + linear model

事实上，这种regularization不仅可以用在多项式的hypothesis中，还可以应用在logistic regression等其他hypothesis中，都可以达到防止过拟合的效果。

我们目前讨论的多项式是形如$x, x^2, x^3, \cdots, x^n$的形式，若x的范围限定在[-1,1]之间，那么可能导致x^n相对于低阶的值要小得多，则其对于的w非常大，相当于要给高阶项设置很大的惩罚。为了避免出现这种数据大小差别很大的情况，可以使用Legendre Polynomials代替$x, x^2, x^3, \cdots, x^n$这种形式，Legendre Polynomials各项之间是正交的，用它进行多项式拟合的效果更好。关于Legendre Polynomials的概念这里不详细介绍，有兴趣的童鞋可以看一下维基百科。

三、Regularization and VC Theory

下面我们研究一下Regularization与VC理论之间的关系。Augmented Error表达式如下：

$$E_{aug}(w) = E_{in}(w) + \frac{\lambda}{N} w^T w$$

VC Bound表示为：

$$E_{out}(w) \le E_{in}(w) + \Omega(H)$$

其中$w^T w$表示的是单个hypothesis的复杂度，记为$\Omega(w)$；而$\Omega(H)$表示整个hypothesis set的复杂度。根据Augmented Error和VC Bound的表达式，$\Omega(w)$包含于$\Omega(H)$之内，所以，$E_{aug}(w)$比E_{in}更接近于E_{out}，即更好地代表E_{out}，$E_{aug}(w)$与E_{out}之间的误差更小。



Augmented Error

$E_{aug}(\mathbf{w}) = E_{in}(\mathbf{w}) + \frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$

VC Bound

$E_{out}(\mathbf{w}) \le E_{in}(\mathbf{w}) + \Omega(\mathcal{H})$

- regularizer $\mathbf{w}^T\mathbf{w}$: complexity of a single hypothesis
- generalization price $\Omega(\mathcal{H})$: complexity of a hypothesis set
- if $\frac{\lambda}{N}\Omega(\mathbf{w})$ 'represents' $\Omega(\mathcal{H})$ well,
  E_{aug} is a better proxy of E_{out} than E_{in}

根据VC Dimension理论，整个hypothesis set的$d_{VC} = \check{d} + 1$，这是因为所有的w都考虑了，没有任何限制条件。而引入限定条件的$d_{VC}(H(C)) = d_{EFF}(H, A)$，即有效的VC dimension。也就是说，$d_{VC}(H)$比较大，因为它代表了整个hypothesis set，但是$d_{EFF}(H, A)$比较小，因为由于regularized的影响，限定了w只取一小部分。其中A表示regularized算法。当$\lambda > 0$时，有：

$$d_{EFF}(H, A) \le d_{VC}$$

这些与实际情况是相符的，比如对多项式拟合模型，当$\lambda = 0$时，所有的w都给予考虑，相应的d_{VC}很大，容易发生过拟合。当$\lambda > 0$且越来越大时，很多w将被舍弃，$d_{EFF}(H, A)$减小，拟合曲线越来越平滑，容易发生欠拟合。

四、General Regularizers

那么通用的Regularizers，即$\Omega(w)$，应该选择什么样的形式呢？一般地，我们会朝着目标函数的方向进行选取。有三种方式：

- **target-dependent**
- **plausible**
- **friendly**

- target-dependent: some properties of target, if known
  - symmetry regularizer: $\sum [\![q \text{ is odd}]\!]\, w_q^2$
- plausible: direction towards smoother or simpler
  stochastic/deterministic noise both **non-smooth**
  - sparsity (L1) regularizer: $\sum |w_q|$ (next slide)
- friendly: easy to optimize
  - weight-decay (L2) regularizer: $\sum w_q^2$



其实这三种方法跟之前error measure类似，其也有三种方法：

- **user-dependent**
- **plausible**
- **friendly**

regularizer与error measure是机器学习模型设计中的重要步骤。

augmented error = error $\widehat{\text{err}}$ + regularizer Ω
regularizer: target-dependent, plausible, or friendly
ringing a bell? :-)
error measure: user-dependent, plausible, or friendly

接下来，介绍两种Regularizer：L2和L1。L2 Regularizer一般比较通用，其形式如下：

$$\Omega(w)=\sum_{q=0}^{Q}w_q^2=||w||_2^2$$

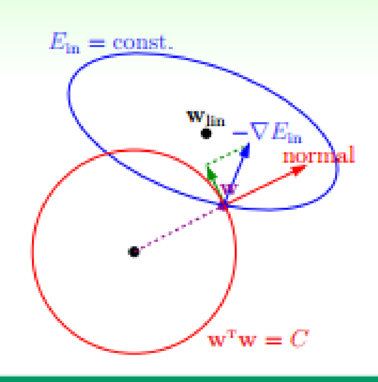
这种形式的regularizer计算的是w的平方和，是凸函数，比较平滑，易于微分，容易进行最优化计算。

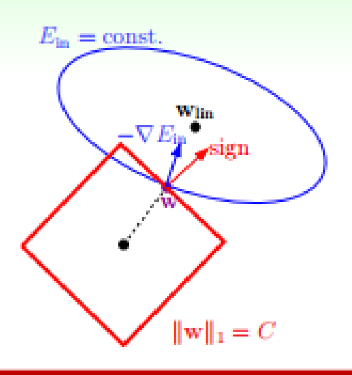
L1 Regularizer的表达式如下：

$$\Omega(w)=\sum_{q=0}^{Q}|w_q|=||w||_1$$

L1计算的不是w的平方和，而是绝对值和，即长度和，也是凸函数。已知$w^Tw=C$围成的是圆形，而$||w||_1=C$围成的是正方形，那么在正方形的四个顶点处，是不可微分的（不像圆形，处处可微分）。根据之前介绍的平行等式推导过程，对应这种正方形，它的解大都位于四个顶点处（不太理解，欢迎补充赐教），因为正方形边界处的w绝对值都不为零，若$-\nabla E_{in}$不与其平行，那么w就会向顶点处移动，顶点处的许多w分量为零，所以，L1 Regularizer的解是稀疏的，称为sparsity。优点是计算速度快。







L2 Regularizer

$$\Omega(\mathbf{w}) = \sum_{q=0}^{Q} w_q^2 = \|\mathbf{w}\|_2^2$$

- convex, differentiable everywhere
- easy to optimize

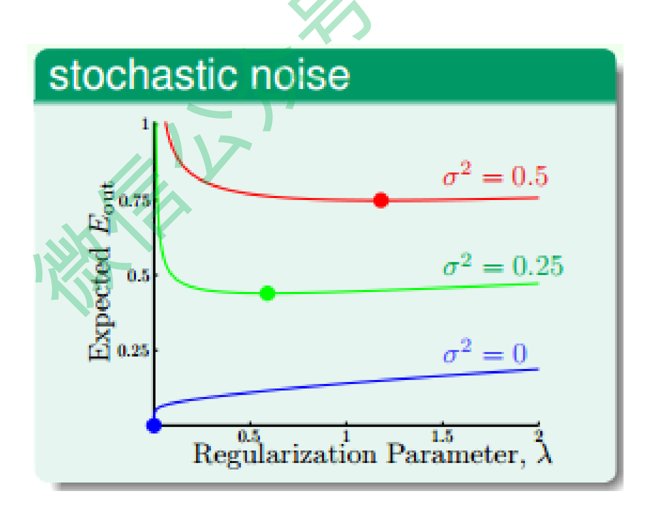
L1 Regularizer

$$\Omega(\mathbf{w}) = \sum_{q=0}^{Q} |w_q| = \|\mathbf{w}\|_1$$

- convex, not differentiable everywhere
- sparsity in solution

L1 useful if needing sparse solution

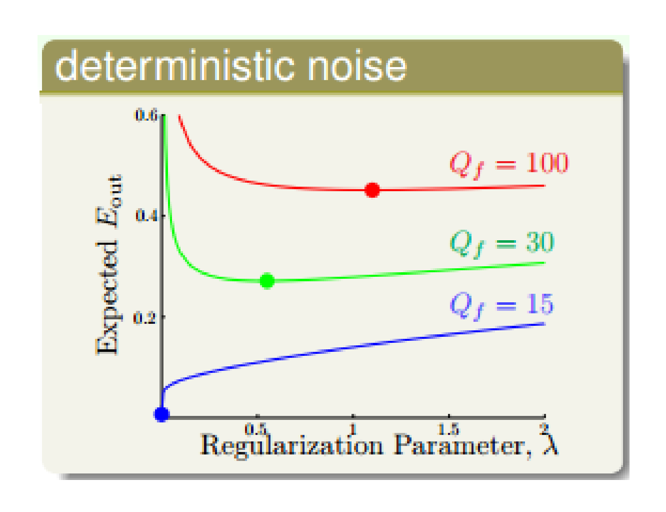
下面来看一下λ如何取值，首先，若stochastic noise不同，那么一般情况下，λ取值有如下特点：



从图中可以看出，stochastic noise越大，λ越大。

另一种情况，不同的deterministic noise，λ取值有如下特点：





从图中可以看出，deterministic noise越大，λ越大。

以上两种noise的情况下，都是noise越大，相应的λ也就越大。这也很好理解，如果在开车的情况下，路况也不好，即noise越多，那么就越会踩刹车，这里踩刹车指的就是regularization。但是大多数情况下，noise是不可知的，这种情况下如何选择λ？这部分内容，我们下节课将会讨论。

五、总结

本节课主要介绍了Regularization。首先，原来的hypothesis set加上一些限制条件，就成了Regularized Hypothesis Set。加上限制条件之后，我们就可以把问题转化为E_{aug}最小化问题，即把w的平方加进去。这种过程，实际上回降低VC Dimension。最后，介绍regularization是通用的机器学习工具，设计方法通常包括target-dependent，plausible，friendly等等。下节课将介绍如何选取合适的λ来建立最佳拟合模型。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程



林轩田《机器学习基石》课程笔记15 -- Validation

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要讲了为了避免overfitting，可以使用regularization方法来解决。在之前的E_{in}上加上一个regularizer，生成E_{aug}，将其最小化，这样可以有效减少模型的复杂度，避免过拟合现象的发生。那么，机器学习领域还有许多选择，如何保证训练的模型具有良好的泛化能力？本节课将介绍一些概念和方法来解决这个选择性的问题。

一、Model Selection Problem

机器学习模型建立的过程中有许多选择，例如对于简单的二元分类问题，首先是算法A的选择，有PLA，pocket，linear regression，logistic regression等等；其次是迭代次数T的选择，有100，1000,10000等等；之后是学习速率η的选择，有1，0.01,0.0001等等；接着是模型特征转换Φ的选择，有linear，quadratic，poly-10，Legendre-poly-10等等；然后是正则化regularizer的选择，有L2，L1等等；最后是正则化系数λ的选择，有0，0.01，1等等。不同的选择搭配，有不同的机器学习效果。我们的目标就是找到最合适的选择搭配，得到一个好的矩g，构建最佳的机器学习模型。

Even Just for Binary Classification . . .

$\mathcal{A} \in$ { PLA, pocket, linear regression, logistic regression}

×

$T \in$ { 100, 1000, 10000}

×

$\eta \in$ { 1, 0.01, 0.0001}

×

$\mathbf{\Phi} \in$ { linear, quadratic, poly-10, Legendre-poly-10}

×

$\Omega(\mathbf{w}) \in$ { L2 regularizer, L1 regularizer, symmetry regularizer}

×

$\lambda \in$ { 0, 0.01, 1 }

假设有M个模型，对应有$H_1, H_2, \cdots, H_M$，即有M个hypothesis set，演算法为$A_1, A_2, \cdots, A_M$，共M个。我们的目标是从这M个hypothesis set中选择一个模型H_{m^*}，通过演算法A_{m^*}对样本集D的训练，得到一个最好的矩g_{m^*}，使其$E_{out}(g_{m^*})$最小。所以，问题的关键就是机器学习中如何选择到最好的矩g_{m^*}。



考虑有这样一种方法，对M个模型分别计算使E_{in}最小的矩g，再横向比较，取其中能使E_{in}最小的模型的矩g_{m^*}：

$$m^* = \underset{1\le m\le M}{\operatorname{argmin}}(E_m = E_{\text{in}}(\mathcal{A}_m(\mathcal{D})))$$

但是E_{in}足够小并不能表示模型好，反而可能表示训练的矩g_{m^*}发生了过拟合，泛化能力很差。而且这种“模型选择+学习训练”的过程，它的VC Dimension是$d_{VC}(H_1 \cup H_2)$，模型复杂度增加。总的来说，泛化能力差，用E_{in}来选择模型是不好的。

另外一种方法，如果有这样一个独立于训练样本的测试集，将M个模型在测试集上进行测试，看一下E_{test}的大小，则选取E_{test}最小的模型作为最佳模型：

$$m^* = \underset{1\le m\le M}{\operatorname{argmin}}(E_m = E_{\text{test}}(\mathcal{A}_m(\mathcal{D})))$$

这种测试集验证的方法，根据finite-bin Hoffding不等式，可以得到：

$$E_{out}(g_{m^*}) \le E_{test}(g_{m^*}) + O(\sqrt{\frac{logM}{N_{test}}})$$

由上式可以看出，模型个数M越少，测试集数目越大，那么$O(\sqrt{\frac{logM}{N_{test}}})$越小，即$E_{test}(g_{m^*})$越接近于$E_{out}(g_{m^*})$。

下面比较一下之前讲的两种方法，第一种方法使用E_{in}作为判断基准，使用的数据集就是训练集D本身；第二种方法使用E_{test}作为判断基准，使用的是独立于训练集D之外的测试集。前者不仅使用D来训练不同的g_m，而且又使用D来选择最好的g_{m^*}，那么g_{m^*}对未知数据并不一定泛化能力好。举个例子，这相当于老师用学生做过的练习题再来对学生进行考试，那么即使学生得到高分，也不能说明他的学习能力强。所以最小化E_{in}的方法并不科学。而后者使用的是独立于D的测试集，相当于新的考试题能更好地反映学生的真实水平，所以最小化E_{test}更加理想。



in-sample error E_{in}

- calculated from $\mathcal{D}$
- feasible on hand
- 'contaminated' as $\mathcal{D}$ also used by $\mathcal{A}_m$ to 'select' g_m

test error E_{test}

- calculated from $\mathcal{D}_{test}$
- infeasible in boss's safe
- 'clean' as $\mathcal{D}_{test}$ never used for selection before

但是，我们拿到的一都是训练集D，测试集是拿不到的。所以，寻找一种折中的办法，我们可以使用已有的训练集D来创造一个验证集validation set，即从D中划出一部分D_{val}作为验证集。D另外的部分作为训练模型使用，D_{val}独立开来，用来测试各个模型的好坏，最小化E_{val}，从而选择最佳的g_{m^*}。

something in between: E_{val}

- calculated from $\mathcal{D}_{val} \subset \mathcal{D}$
- feasible on hand
- 'clean' if $\mathcal{D}_{val}$ never used by $\mathcal{A}_m$ before

二、Validation

从训练集D中抽出一部分K个数据作为验证集D_{val}，D_{val}对应的error记为E_{val}。这样做的一个前提是保证D_{val}独立同分布（iid）于P(x,y)，也就是说D_{val}的选择是从D中平均随机抽样得到的，这样能够把E_{val}与E_{out}联系起来。D中去除D_{val}后的数据就是供模型选择的训练数据D_{train}，其大小为N-k。从D_{train}中选择最好的矩，记为g_m^-。

$$\begin{array}{ccccc} E_{in}(h) & & & & E_{val}(h) \\ \uparrow & & & & \uparrow \\ \underbrace{\mathcal{D}}_{\text{size } N} & \rightarrow & \underbrace{\mathcal{D}_{train}}_{\text{size } N-K} & \cup & \underbrace{\mathcal{D}_{val}}_{\text{size } K} \\ \downarrow & & \downarrow & & \\ g_m = \mathcal{A}_m(\mathcal{D}) & & g_m^- = \mathcal{A}_m(\mathcal{D}_{train}) & & \end{array}$$

假如D共有1000个样本，那么可以选择其中900个D_{train}，剩下的100个作为D_{val}。使用D_{train}训练模型，得到最佳的g_m^-，使用g_m^-对D_{val}进行验证，得到如下Hoffding不等式：



$$E_{out}(g_m^-) \leq E_{val}(g_m^-) + O(\sqrt{\frac{logM}{K}})$$

假设有M种模型hypothesis set，D_{val}的数量为K，那么从每种模型m中得到一个在D_{val}上表现最好的矩，再横向比较，从M个矩中选择一个最好的$m*$作为我们最终得到的模型。

$$m^* = \underset{1 \leq m \leq M}{\text{argmin}}(E_m = E_{\text{val}}(\mathcal{A}_m(\mathcal{D}_{\text{train}})))$$

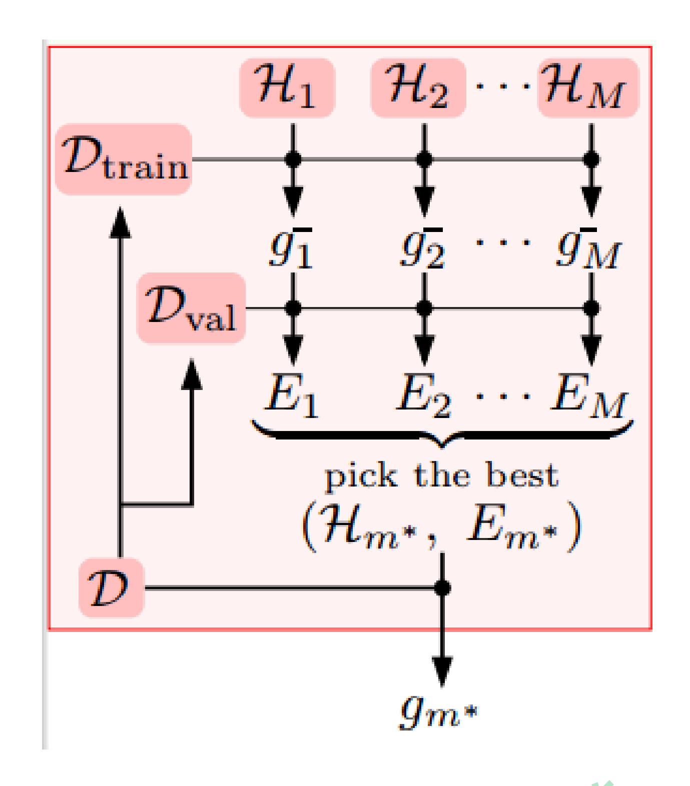
现在由于数量为N的总样本D的一部分K作为验证集，那么只有N-k个样本可供训练。从D_{train}中得到最好的$g_{m^*}^-$，而总样本D对应的最好的矩为g_{m^*}。根据之前的leraning curve很容易知道，训练样本越多，得到的模型越准确，其hypothesis越接近target function，即D的E_{out}比D_{train}的E_{out}要小：

$$E_{\text{out}}\Big(\underbrace{g_{m^*}}_{\mathcal{A}_{m^*}(\mathcal{D})}\Big) \leq E_{\text{out}}\Big(\underbrace{g_{m^*}^-}_{\mathcal{A}_{m^*}(\mathcal{D}_{\text{train}})}\Big)$$

所以，我们通常的做法是通过D_{val}来选择最好的矩$g_{m^*}^-$对应的模型m^*，再对整体样本集D使用该模型进行训练，最终得到最好的矩g_{m^*}。

总结一下，使用验证集进行模型选择的整个过程为：先将D分成两个部分，一个是训练样本D_{train}，一个是验证集D_{val}。若有M个模型，那么分别对每个模型在D_{train}上进行训练，得到矩g_m^-，再用D_{val}对每个g_m^-进行验证，选择表现最好的矩$g_{m^*}^-$，则该矩对应的模型被选择。最后使用该模型对整个D进行训练，得到最终的g_{m^*}。下图展示了整个模型选择的过程：

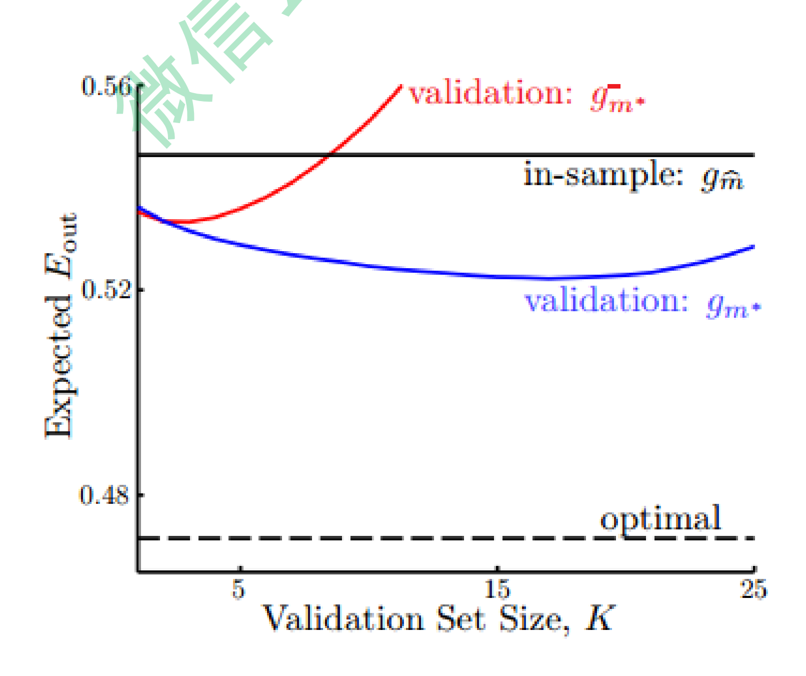




不等式关系满足：

$$E_{out}(g_{m^*}) \leq E_{out}(g_{m^*}^-) \leq E_{val}(g_{m^*}^-) + O(\sqrt{\frac{logM}{K}})$$

下面我们举个例子来解释这种模型选择的方法的优越性，假设有两个模型：一个是5阶多项式H_{Φ_5}，一个是10阶多项式$H_{\Phi_{10}}$。通过不使用验证集和使用验证集两种方法对模型选择结果进行比较，分析结果如下：



图中，横坐标表示验证集数量K，纵坐标表示E_{out}大小。黑色水平线表示没有验证集，完全使用E_{in}进行判断基准，那么$H_{\Phi_{10}}$更好一些，但是这种方法的E_{out}比较大，而且与K无关。黑色虚线表示测试集非常接近实际数据，这是一种理想的情况，其



E_{out}很小，同样也与K无关，实际中很难得到这条虚线。红色曲线表示使用验证集，但是最终选取的矩是$g_{m^*}^-$，其趋势是随着K的增加，它对应的E_{out}先减小再增大，当K大于一定值的时候，甚至会超过黑色水平线。蓝色曲线表示也使用验证集，最终选取的矩是g_{m^*}，其趋势是随着K的增加，它对应的E_{out}先缓慢减小再缓慢增大，且一直位于红色曲线和黑色直线之下。从此可见，蓝色曲线对应的方法最好，符合我们之前讨论的使用验证集进行模型选择效果最好。

这里提一点，当K大于一定的值时，红色曲线会超过黑色直线。这是因为随着K的增大，D_{val}增大，但可供模型训练的D_{train}在减小，那得到的$g_{m^*}^-$不具有很好的泛化能力，即对应的E_{out}会增大，甚至当K增大到一定值时，比E_{in}模型更差。

那么，如何设置验证集K值的大小呢？根据之前的分析：

$$E_{out}(g) \underset{\text{(small } K)}{\approx} E_{out}(g^-) \underset{\text{(large } K)}{\approx} E_{val}(g^-)$$

当K值很大时，$E_{val} \approx E_{out}$，但是g_m^-与g_m相差很大；当K值很小是，$g_m^- \approx g_m$，但是E_{val}与E_{out}可能相差很大。所以有个折中的办法，通常设置$k = \frac{N}{5}$。值得一提的是，划分验证集，通常并不会增加整体时间复杂度，反而会减少，因为D_{train}减少了。

三、Leave-One-Out Cross Validation

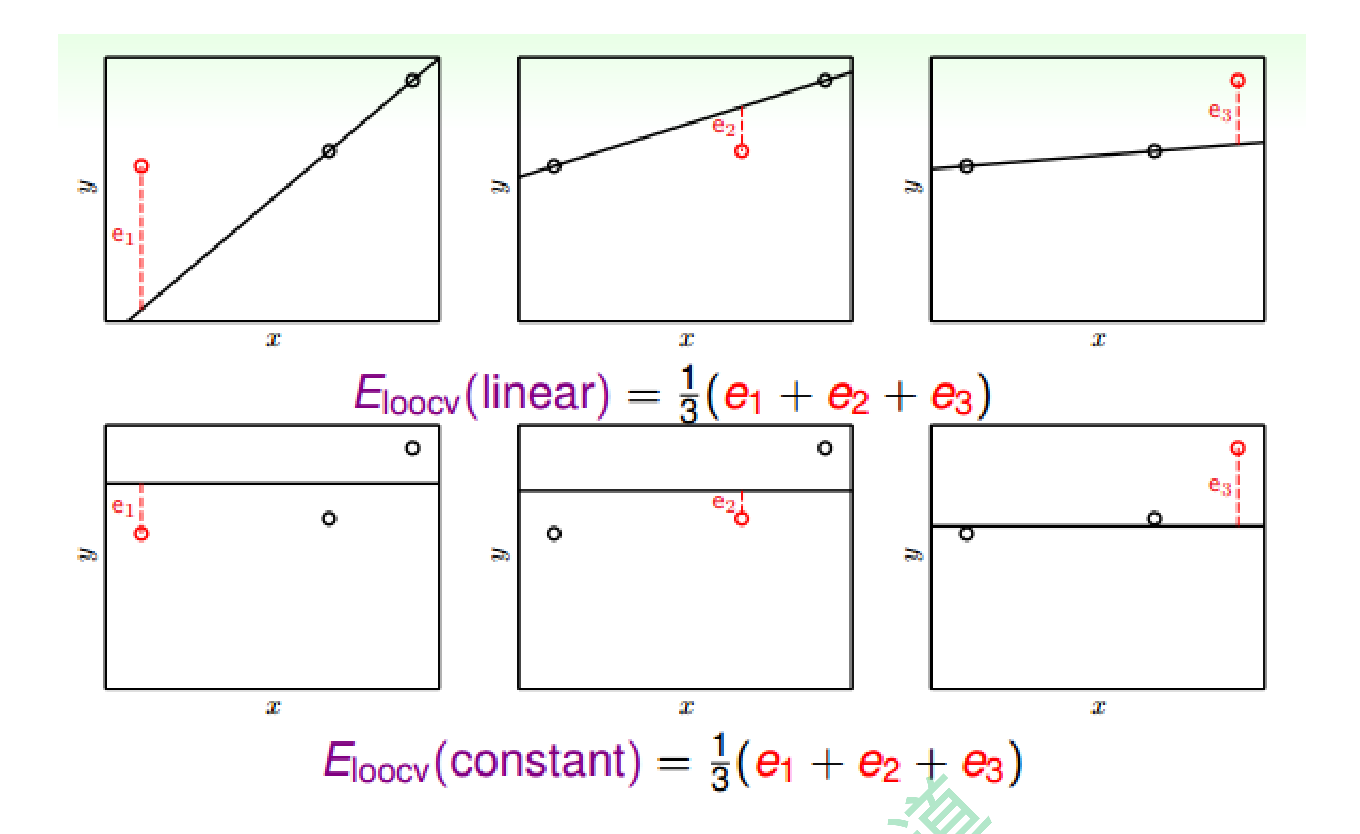
假如考虑一个极端的例子，k=1，也就是说验证集大小为1，即每次只用一组数据对g_m进行验证。这样做的优点是$g_m^- \approx g_m$，但是E_{val}与E_{out}可能相差很大。为了避免E_{val}与E_{out}相差很大，每次从D中取一组作为验证集，直到所有样本都作过验证集，共计算N次，最后对验证误差求平均，得到$E_{loocv}(H,A)$，这种方法称之为留一法交叉验证，表达式为：

$$E_{loocv}(H,A) = \frac{1}{N}\sum_{n=1}^{N} e_n = \frac{1}{N}\sum_{n=1}^{N} err(g_n^-(x_n), y_n)$$

这样求平均的目的是为了让$E_{loocv}(H,A)$尽可能地接近$E_{out}(g)$。

下面用一个例子图解留一法的过程：





如上图所示，要对二维平面上的三个点做拟合，上面三个图表示的是线性模型，下面三个图表示的是常数模型。对于两种模型，分别使用留一交叉验证法来计算E_{loocv}，计算过程都是每次将一个点作为验证集，其他两个点作为训练集，最终将得到的验证误差求平均值，就得到了$E_{loocv}(linear)$和$E_{loocv}(constant)$，比较两个值的大小，取值小对应的模型即为最佳模型。

$$m^* = \underset{1 \le m \le M}{\operatorname{argmin}}(E_m = E_{loocv}(\mathcal{H}_m, \mathcal{A}_m))$$

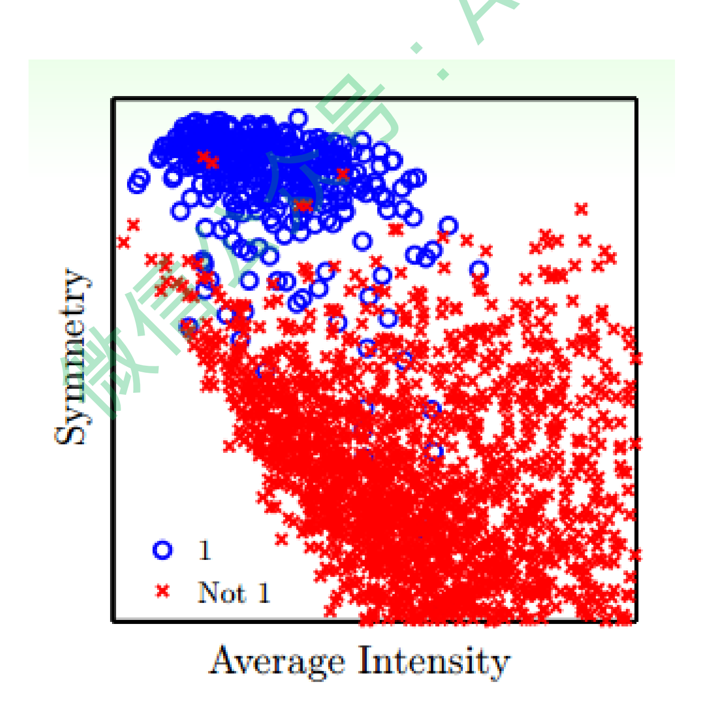
接下来，我们从理论上分析Leave-One-Out方法的可行性，即$E_{loocv}(H,A)$是否能保证E_{out}的矩足够好？假设有不同的数据集D，它的期望分布记为ε_D，则其$E_{loocv}(H,A)$可以通过推导，等于$E_{out}(N-1)$的平均值。由于N-1近似为N，$E_{out}(N-1)$的平均值也近似等于$E_{out}(N)$的平均值。具体推导过程如下：



$$
\begin{aligned}
\mathop{\mathcal{E}}_{\mathcal{D}} E_{\text{loocv}}(\mathcal{H}, \mathcal{A}) = \mathop{\mathcal{E}}_{\mathcal{D}} \frac{1}{N}\sum_{n=1}^{N} e_n &= \frac{1}{N}\sum_{n=1}^{N} \mathop{\mathcal{E}}_{\mathcal{D}} e_n \\
&= \frac{1}{N}\sum_{n=1}^{N} \mathop{\mathcal{E}}_{\mathcal{D}_n} \mathop{\mathcal{E}}_{(\mathbf{x}_n, y_n)} \text{err}(g_n^-(\mathbf{x}_n), y_n) \\
&= \frac{1}{N}\sum_{n=1}^{N} \mathop{\mathcal{E}}_{\mathcal{D}_n} E_{\text{out}}(g_n^-) \\
&= \frac{1}{N}\sum_{n=1}^{N} \overline{E_{\text{out}}}(N-1) = \overline{E_{\text{out}}}(N-1)
\end{aligned}
$$

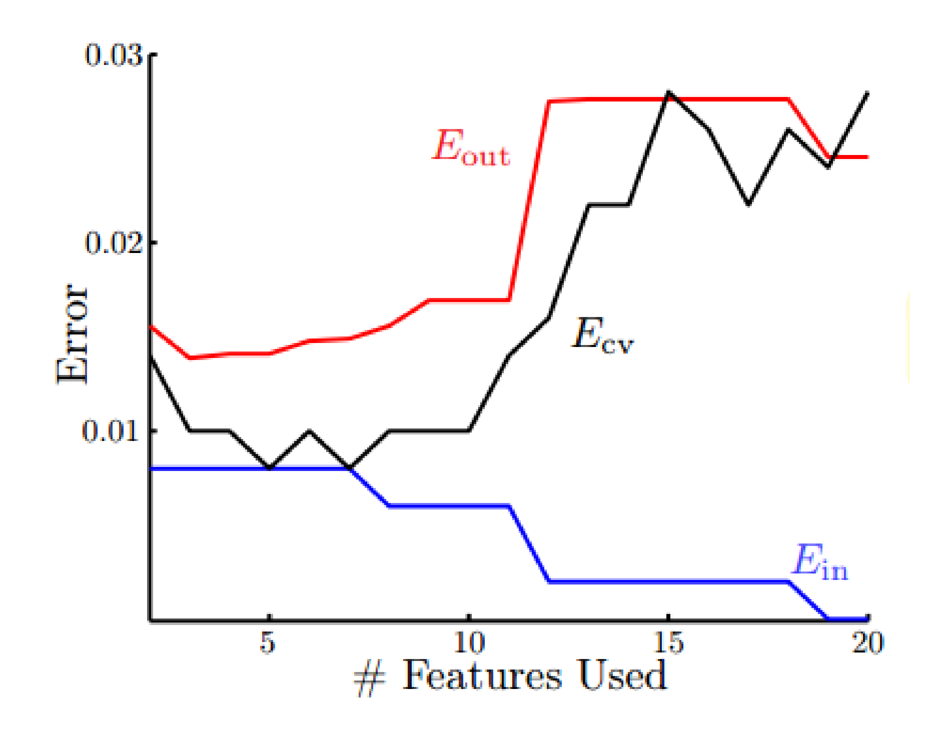
最终我们得到的结论是$E_{loocv}(H, A)$的期望值和$E_{out}(g^-)$的期望值是相近的，这代表得到了比较理想的$E_{out}(g)$，Leave-One-Out方法是可行的。

举一个例子，使用两个特征：Average Intensity和Symmetry加上这两个特征的非线性变换（例如高阶项）来进行手写数字识别。平面特征分布如下图所示：

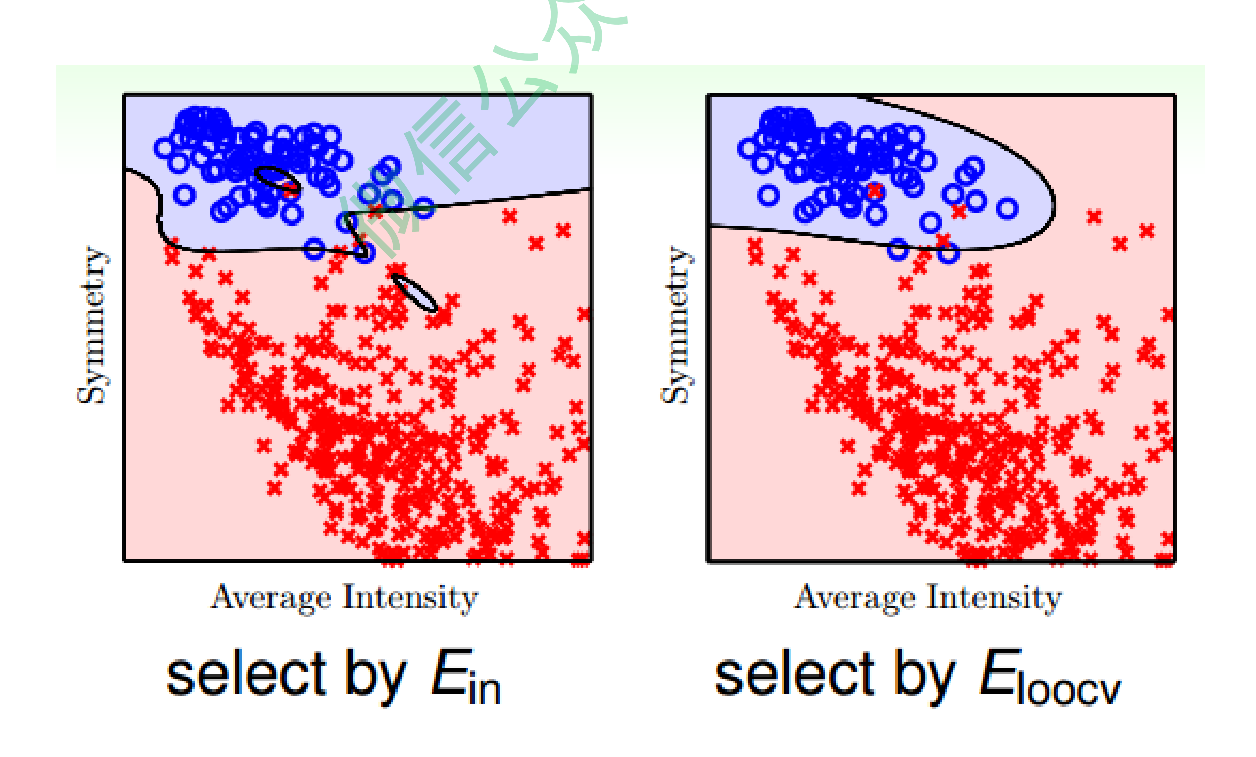


Error与特征数量的关系如下图所示：





从图中我们看出，随着特征数量的增加，E_{in}不断减小，E_{out}先减小再增大，虽然E_{in}是不断减小的，但是它与E_{out}的差距越来越大，发生了过拟合，泛化能力太差。而E_{cv}与E_{out}的分布基本一致，能较好地反映E_{out}的变化。所以，我们只要使用Leave-One-Out方法得到使E_{cv}最小的模型，就能保证其E_{out}足够小。下图是分别使用E_{in}和E_{out}进行训练得到的分类曲线：



很明显可以看出，使用E_{in}发生了过拟合，而E_{loocv}分类效果更好，泛化能力强。

四、V-Fold Cross Validation



接下来我们看看Leave-One-Out可能的问题是什么。首先，第一个问题是计算量，假设N=1000，那么就需要计算1000次的E_{loocv}，再计算其平均值。当N很大的时候，计算量是巨大的，很耗费时间。第二个问题是稳定性，例如对于二分类问题，取值只有0和1两种，预测本身存在不稳定的因素，那么对所有的E_{loocv}计算平均值可能会带来很大的数值跳动，稳定性不好。所以，这两个因素决定了Leave-One-Out方法在实际中并不常用。

针对Leave-One-Out的缺点，我们对其作出了改进。Leave-One-Out是将N个数据分成N分，那么改进措施是将N个数据分成V份（例如V=10），计算过程与Leave-One-Out相似。这样可以减少总的计算量，又能进行交叉验证，得到最好的矩，这种方法称为V-折交叉验证。其实Leave-One-Out就是V-折交叉验证的一个极端例子。

$$E_{cv}(H,A)=\frac{1}{V}\sum_{v=1}^{V}E_{val}^{(V)}(g_V^-)$$

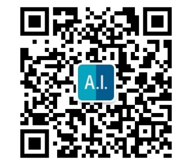
所以呢，一般的Validation使用V-折交叉验证来选择最佳的模型。值得一提的是Validation的数据来源也是样本集中的，所以并不能保证交叉验证的效果好，它的模型一定好。只有样本数据越多，越广泛，那么Validation的结果越可信，其选择的模型泛化能力越强。

五、总结

本节课主要介绍了Validation验证。先从如何选择一个好的模型开始切入，例如使用E_{in}、E_{test}都是不太好的，最终使用E_{val}来进行模型选择。然后详细介绍了Validation的过程。最后，介绍了Leave-One-Out和V-Fold Cross两种验证方法，比较它们各自的优点和缺点，实际情况下，V-Fold Cross更加常用。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程



林轩田《机器学习基石》课程笔记16（完结） -- Three Learning Principles

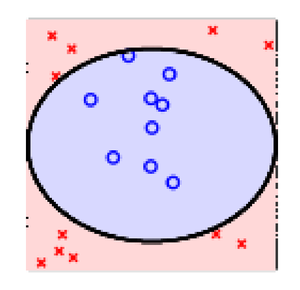
作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们讲了一个机器学习很重要的工具——Validation。我们将整个训练集分成两部分：D_{train}和D_{val}，一部分作为机器学习模型建立的训练数据，另一部分作为验证模型好坏的数据，从而选择到更好的模型，实现更好的泛化能力。这节课，我们主要介绍机器学习中非常实用的三个“锦囊妙计”。

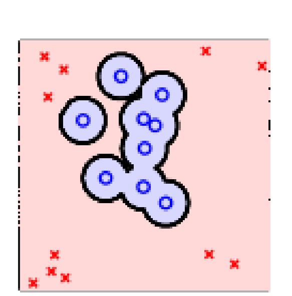
一、Occam's Razor

奥卡姆剃刀定律（Occam's Razor），是由14世纪逻辑学家、圣方济各会修士奥卡姆的威廉（William of Occam，约1285年至1349年）提出。奥卡姆（Ockham）在英格兰的萨里郡，那是他出生的地方。他在《箴言书注》2卷15题说“切勿浪费较多东西去做用较少的东西同样可以做好的事情。” 这个原理称为“如无必要，勿增实体”（Entities must not be multiplied unnecessarily），就像剃刀一样，将不必要的部分去除掉。

Occam's Razor反映到机器学习领域中，指的是在所有可能选择的模型中，我们应该选择能够很好地解释已知数据并且十分简单的模型。



which one do you prefer? :-)



上图就是一个模型选择的例子，左边的模型很简单，可能有分错的情况；而右边的模型非常复杂，所有的训练样本都分类正确。但是，我们会选择左边的模型，它更简单，符合人类直觉的解释方式。这样的结果带来两个问题：一个是什么模型称得上是简单的？另一个是为什么简单模型比复杂模型要好？

简单的模型一方面指的是简单的hypothesis h，简单的hypothesis就是指模型使用的



征比较少，例如多项式阶数比较少。简单模型另一方面指的是模型H包含的hypothesis数目有限，不会太多，这也是简单模型包含的内容。

simple hypothesis h	simple model $\mathcal{H}$
• small $\Omega(h)$ = 'looks' simple • specified by **few parameters**	• small $\Omega(\mathcal{H})$ = not many • contains **small number of hypotheses**

其实，simple hypothesis h和simple model H是紧密联系的。如果hypothesis的特征个数是l，那么H中包含的hypothesis个数就是2^l，也就是说，hypothesis特征数目越少，H中hypothesis数目也就越少。

所以，为了让模型简单化，我们可以一开始就选择简单的model，或者用regularization，让hypothesis中参数个数减少，都能降低模型复杂度。

那为什么简单的模型更好呢？下面从哲学的角度简单解释一下。机器学习的目的是"找规律"，即分析数据的特征，总结出规律性的东西出来。假设现在有一堆没有规律的杂乱的数据需要分类，要找到一个模型，让它的$E_{in}=0$，是很难的，大部分时候都无法正确分类，但是如果是很复杂的模型，也有可能将其分开。反过来说，如果有另一组数据，如果可以比较容易找到一个模型能完美地把数据分开，那表明数据本身应该是有某种规律性。也就是说杂乱的数据应该不可以分开，能够分开的数据应该不是杂乱的。如果使用某种简单的模型就可以将数据分开，那表明数据本身应该符合某种规律性。相反地，如果用很复杂的模型将数据分开，并不能保证数据本身有规律性存在，也有可能是杂乱的数据，因为无论是有规律数据还是杂乱数据，复杂模型都能分开。这就不是机器学习模型解决的内容了。所以，模型选择中，我们应该尽量先选择简单模型，例如最简单的线性模型。

二、Sampling Bias

首先引入一个有趣的例子：1948年美国总统大选的两位热门候选人是Truman和Dewey。一家报纸通过电话采访，统计人们把选票投给了Truman还是Dewey。经过大量的电话统计显示，投给Dewey的票数要比投个Truman的票数多，所以这家报纸就在选举结果还没公布之前，信心满满地发表了"Dewey Defeats Truman"的报纸头版，认为Dewey肯定赢了。但是大选结果公布后，让这家报纸大跌眼镜，最终Truman赢的了大选的胜利。

为什么会出现跟电话统计完全相反的结果呢？是因为电话统计数据出错还是投票运气



不好？都不是。其实是因为当时电话比较贵，有电话的家庭比较少，而正好是有电话的美国人支持Dewey的比较多，而没有电话的支持Truman比较多。也就是说样本选择偏向于有钱人那边，可能不具有广泛的代表性，才造成Dewey支持率更多的假象。

这个例子表明，抽样的样本会影响到结果，用一句话表示“If the data is sampled in a biased way, learning will produce a similarly biased outcome.”意思是，如果抽样有偏差的话，那么学习的结果也产生了偏差，这种情形称之为抽样偏差Sampling Bias。

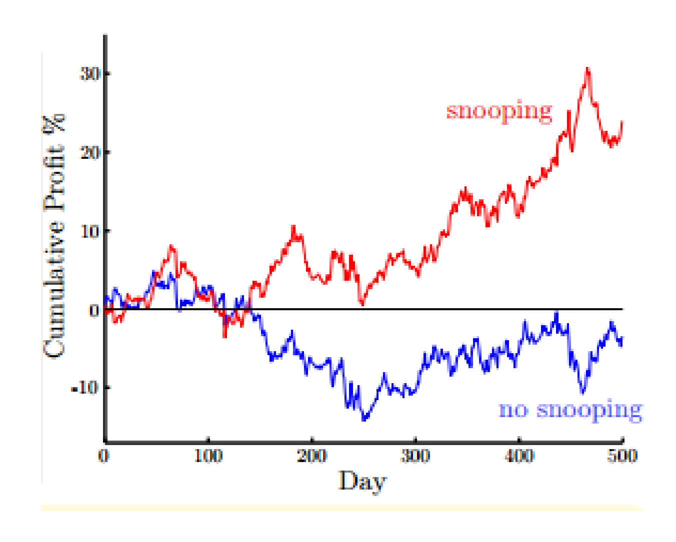
从技术上来说，就是训练数据和验证数据要服从同一个分布，最好都是独立同分布的，这样训练得到的模型才能更好地具有代表性。

三、Data Snooping

之前的课程，我们介绍过在模型选择时应该尽量避免偷窥数据，因为这样会使我们人为地倾向于某种模型，而不是根据数据进行随机选择。所以，Φ应该自由选取，最好不要偷窥到原始数据，这会影响我们的判断。

事实上，数据偷窥发生的情况有很多，不仅仅指我们看到了原始数据。什么意思呢？其实，当你在使用这些数据的任何过程，都是间接地偷看到了数据本身，然后你会进行一些模型的选择或者决策，这就增加了许多的model complexity，也就是引入了污染。

下面举个例子来说明。假如我们有8年的货比交易数据，我们希望从这些数据中找出规律，来预测货比的走势。如果选择前6年数据作为训练数据，后2年数据作为测试数据的话，来训练模型。现在我们有前20天的数据，根据之前训练的模型，来预测第21天的货比交易走势。



现在有两种训练模型的方法，如图所示，一种是使用前6年数据进行模型训练，后2年



数据作为测试，图中蓝色曲线表示后2年的预测收益；另一种是直接使用8年数据进行模型训练，图中红色曲线表示后2年的预测收益情况。图中，很明显，使用8年数据进行训练的模型对后2年的预测的收益更大，似乎效果更好。但是这是一种自欺欺人的做法，因为训练的时候已经拿到了后2年的数据，用这样的模型再来预测后2年的走势是不科学的。这种做法也属于间接偷窥数据的行为。直接偷窥和间接偷窥数据的行为都是不科学的做法，并不能表示训练的模型有多好。

- snooping: shift-scale all values by training + testing
- no snooping: shift-scale all values by training only

还有一个偷窥数据的例子，比如对于某个基准数据集D，某人对它建立了一个模型H1，并发表了论文。第二个人看到这篇论文后，又会对D，建立一个新的好的模型H2。这样，不断地有人看过前人的论文后，建立新的模型。其实，后面人选择模型时，已经被前人影响了，这也是偷窥数据的一种情况。也许你能对D训练很好的模型，但是可能你仅仅只根据前人的模型，成功避开了一些错误，甚至可能发生了overfitting或者bad generalization。所以，机器学习领域有这样一句有意思的话“If you torture the data long enough, it will confess.”所以，我们不能太“折磨”我们的数据了，否则它只能“妥协”了~哈哈。

在机器学习过程中，避免“偷窥数据”非常重要，但实际上，完全避免也很困难。实际操作中，有一些方法可以帮助我们尽量避免偷窥数据。第一个方法是“看不见”数据。就是说当我们在选择模型的时候，尽量用我们的经验和知识来做判断选择，而不是通过数据来选择。先选模型，再看数据。第二个方法是保持怀疑。就是说时刻保持对别人的论文或者研究成果保持警惕与怀疑，要通过自己的研究与测试来进行模型选择，这样才能得到比较正确的结论。

- be blind: avoid **making modeling decision by data**
- be suspicious: interpret research results (including your own) by proper **feeling of contamination**

四、Power of Three

本小节，我们对16节课做个简单的总结，用“三的威力”进行概括。因为课程中我们介绍的很多东西都与三有关。

首先，我们介绍了跟机器学习相关的三个领域：



- Data Mining
- Artificial Intelligence
- Statistics

Data Mining	Artificial Intelligence	Statistics
• use (huge) data to find property that is interesting • difficult to distinguish ML and DM in reality	• compute something that shows intelligent behavior • ML is one possible route to realize AI	• use data to make inference about an unknown process • statistics contains many useful tools for ML

我们还介绍了三个理论保证：

- Hoeffding
- Multi-Bin Hoeffding
- VC

Hoeffding	Multi-Bin Hoeffding	VC
$P[\text{BAD}] \le 2\exp(-2\epsilon^2 N)$	$P[\text{BAD}] \le 2M\exp(-2\epsilon^2 N)$	$P[\text{BAD}] \le 4m_{\mathcal{H}}(2N)\exp(\ldots)$
• one hypothesis • useful for verifying/testing	• M hypotheses • useful for validation	• all $\mathcal{H}$ • useful for training

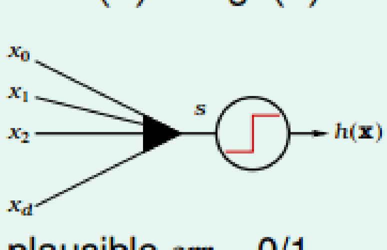
然后，我们又介绍了三种线性模型：

- PLA/pocket
- linear regression
- logistic regression



PLA/pocket

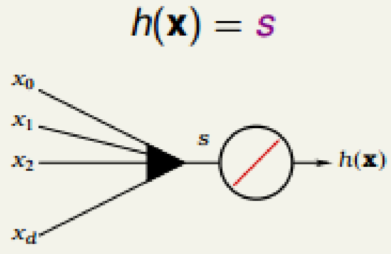
$h(\mathbf{x}) = \text{sign}(s)$



plausible err = 0/1
(small flipping noise)
minimize **specially**

linear regression

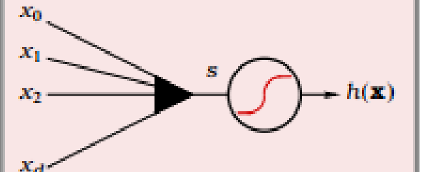
$h(\mathbf{x}) = s$



friendly err = squared
(easy to minimize)
minimize **analytically**

logistic regression

$h(\mathbf{x}) = \theta(s)$



plausible err = CE
(maximum likelihood)
minimize **iteratively**

同时，我们介绍了三种重要的工具：

- Feature Transform
- Regularization
- Validation

Feature Transform

$E_{in}(\mathbf{w}) \rightarrow E_{in}(\tilde{\mathbf{w}})$
$d_{VC}(\mathcal{H}) \rightarrow d_{VC}(\mathcal{H}_{\Phi})$

- by using **more complicated Φ**
- **lower E_{in}**
- higher d_{VC}

Regularization

$E_{in}(\mathbf{w}) \rightarrow E_{in}(\mathbf{w}_{REG})$
$d_{VC}(\mathcal{H}) \rightarrow d_{EFF}(\mathcal{H}, \mathcal{A})$

- by augmenting **regularizer Ω**
- **lower d_{EFF}**
- higher E_{in}

Validation

$E_{in}(h) \rightarrow E_{val}(h)$
$\mathcal{H} \rightarrow \{g_1^-, \ldots, g_M^-\}$

- by reserving K examples as $\mathcal{D}_{val}$
- **fewer choices**
- fewer examples

还有我们本节课介绍的三个锦囊妙计：

- Occam's Razer
- Sampling Bias
- Data Snooping

Occam's Razer
simple is good

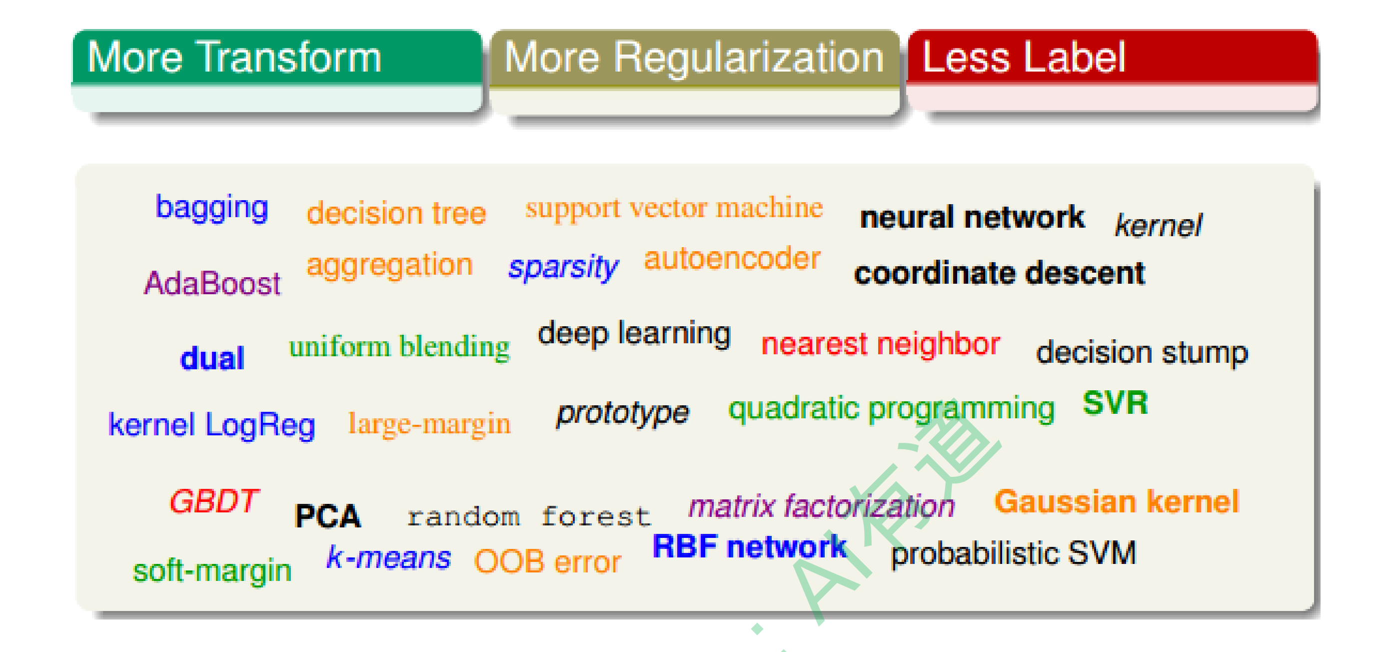
Sampling Bias
class matches exam

Data Snooping
honesty is best policy



最后，我们未来机器学习的方向也分为三种：

- More Transform
- More Regularization
- Less Label



五、总结

本节课主要介绍了机器学习三个重要的锦囊妙计：Occam's Razor, Sampling Bias, Data Snooping。并对《机器学习基石》课程中介绍的所有知识和方法进行"三的威力"这种形式的概括与总结，"三的威力"也就构成了坚固的机器学习基石。

整个机器学习基石的课程笔记总结完毕！后续将会推出机器学习技法的学习笔记，谢谢！

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习基石》课程



林轩田《机器学习技法》课程笔记1 -- Linear Support Vector Machine

作者：红色石头　　公众号：AI有道（id：redstonewill）

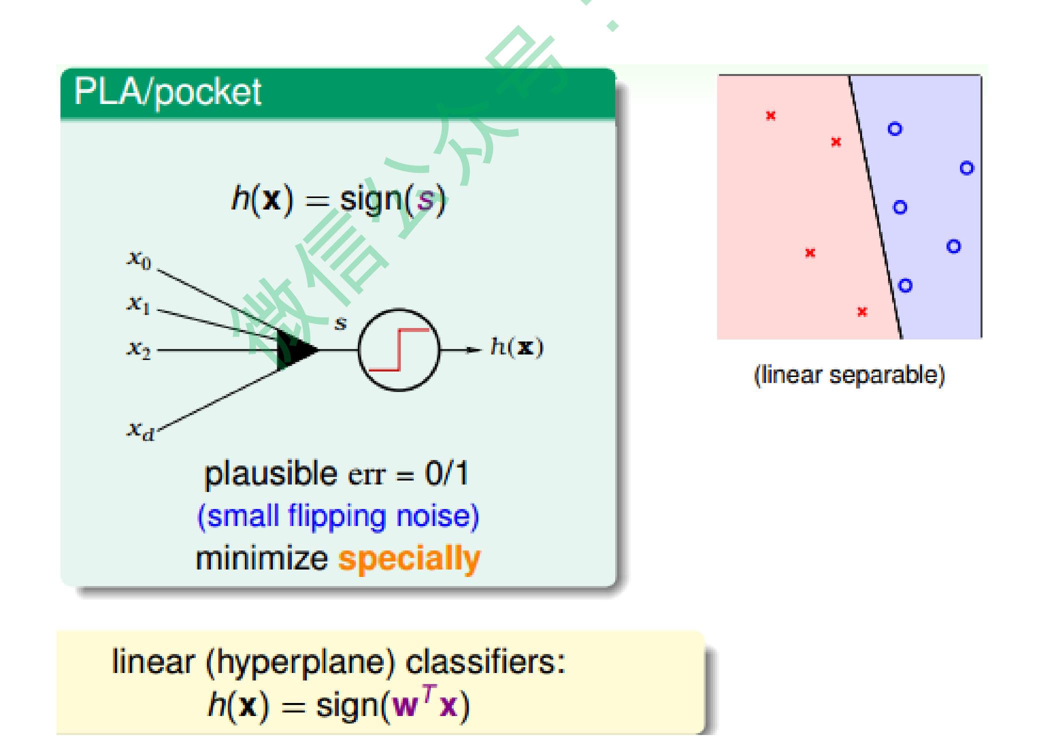
关于台湾大学林轩田老师的《机器学习基石》课程，我们已经总结了16节课的笔记。这里附上基石第一节课的博客地址：

台湾大学林轩田机器学习基石课程学习笔记1 -- The Learning Problem

本系列同样分成16节课，将会介绍《机器学习基石》的进阶版《机器学习技法》，更深入地探讨机器学习一些高级算法和技巧。

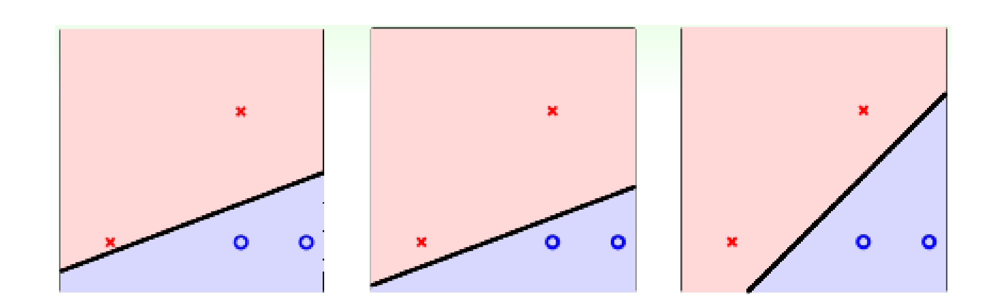
Large-Margin Separating Hyperplane

回顾一下我们之前介绍了linear classification，对于线性可分的情况，我们可以使用PLA/pocket算法在平面或者超平面上把正负类分开。



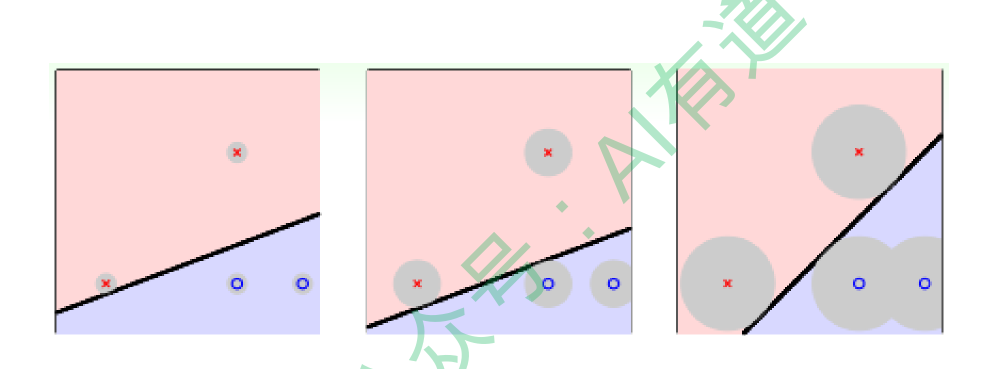
例如对平面2D这种情况，我们可以找到一条直线，能将正类和负类完全分开。但是，这样的直线通常不止一条，如下图所示。那么，下图中的三条分类线都能将数据分开，但是哪条线更好呢？





这三条直线都是由PLA/pocket算法不断修正错误点而最终产生的，整个确定直线形状的过程是随机的。单从分类效果上看，这三条直线都满足要求，而且都满足VC bound要求，模型复杂度$\Omega(H)$是一样的，即具有一定的泛化能力。但是，如果要选择的话，凭第一感觉，我们还是会选择第三条直线，感觉它的分类效果更好一些。那这又是为什么呢？

先给个简单解释，一般情况下，训练样本外的测量数据应该分布在训练样本附近，但与训练样本的位置有一些偏差。若要保证对未知的测量数据也能进行正确分类，最好让分类直线距离正类负类的点都有一定的距离。这样能让每个样本点附近的圆形区域是“安全”的。圆形区域越大，表示分类直线对测量数据误差的容忍性越高，越“安全”。



如上图所示，左边的点距离分类直线的最小距离很小，它的圆形区域很小。那么，这种情况下，分类线对测量数据误差的容忍性就很差，测量数据与样本数据稍有偏差，很有可能就被误分。而右边的点距离分类直线的最小距离更大一些，其圆形区域也比较大。这种情况下，分类线对测量数据误差的容忍性就相对来说大很多，不容易误分。也就是说，左边分类线和右边分类线的最大区别是对这类测量误差的容忍度不同。

那么，如果每一笔训练资料距离分类线越远的话，就表示分类型可以忍受更多的测量误差（noise）。我们之前在《机器学习基石》中介绍过，noise是造成overfitting的主要原因，而测量误差也是一种noise。所以，如果分类线对测量误差的容忍性越好的话，表示这是一条不错的分类线。那么，我们的目标就是找到这样一条最“健壮”的线，即距离数据点越远越好。



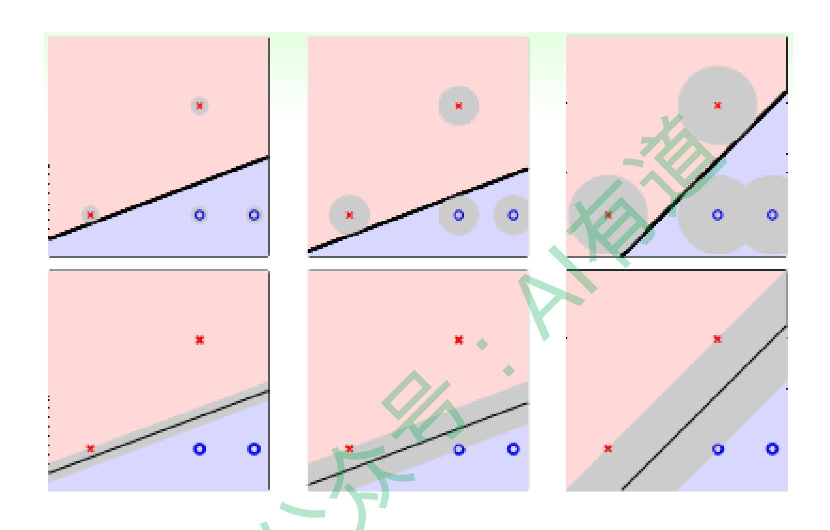
informal argument

if (Gaussian-like) noise on future $\mathbf{x} \approx \mathbf{x}_n$:

$\mathbf{x}_n$ further from hyperplane	distance to closest $\mathbf{x}_n$
$\Longleftrightarrow$ tolerate more noise	$\Longleftrightarrow$ amount of noise tolerance
$\Longleftrightarrow$ more robust to overfitting	$\Longleftrightarrow$ robustness of hyperplane

rightmost one: **more robust**
because of **larger distance to closest $\mathbf{x}_n$**

上面我们用圆形区域表示分类线能够容忍多少误差，也就相当于计算点到直线的距离。距离越大，表示直线越“胖”，越能容忍误差；距离越小，表示直线越“瘦”，越不能容忍误差。越胖越好（像杨贵妃那样的哦~）。



如何定义分类线有多胖，就是看距离分类线最近的点与分类线的距离，我们把它用margin表示。分类线由权重w决定，目的就是找到使margin最大时对应的w值。整体来说，我们的目标就是找到这样的分类线并满足下列条件：

- **分类正确，即$y_n w^T x_n > 0$**
- **margin最大化**

$$\max_{\mathbf{w}} \quad \text{margin}(\mathbf{w})$$
$$\text{subject to} \quad \text{every } y_n \mathbf{w}^T \mathbf{x}_n > 0$$
$$\text{margin}(\mathbf{w}) = \min_{n=1,\ldots,N} \text{distance}(\mathbf{x}_n, \mathbf{w})$$

Standard Large-Margin Problem

要让margin最大，即让离分类线最近的点到分类线距离最大，我们先来看一下如何计算点到分类线的距离。

首先，我们将权重$w(w_0, w_1, \cdots, w_d)$中的w_0拿出来，用b表示。同时省去x_0项。这样，



hypothesis就变成了$h(x) = sign(w^T x + b)$。

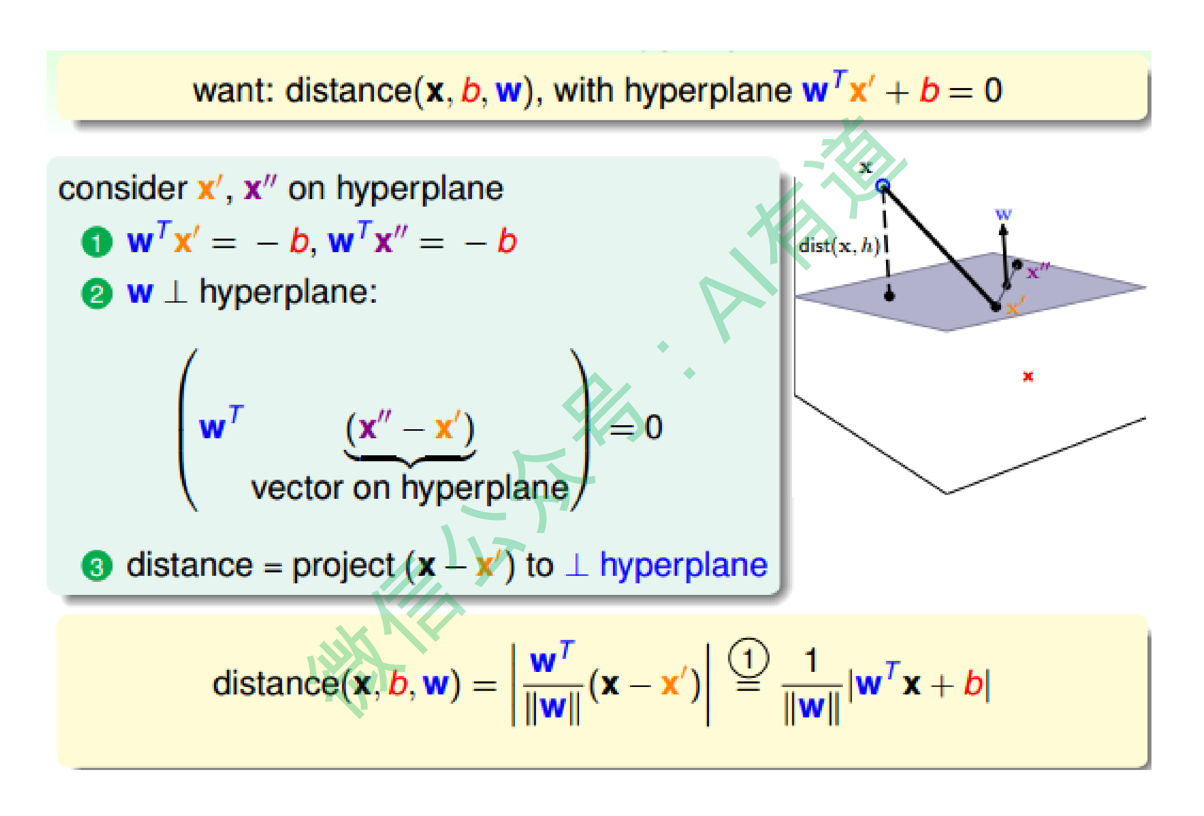
'shorten' **x** and **w**

distance needs w_0 and $(w_1, \ldots, w_d)$ differently (to be derived)

$$b = w_0 \qquad \cancel{x_0 = 1}$$

$$\begin{bmatrix} | \\ \mathbf{w} \\ | \end{bmatrix} = \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}; \quad \begin{bmatrix} | \\ \mathbf{x} \\ | \end{bmatrix} = \begin{bmatrix} x_1 \\ \vdots \\ x_d \end{bmatrix}$$

下面，利用图解的方式，详细推导如何计算点到分类平面的距离：



如上图所示，平面上有两个点：x'和x"。因为这两个点都在分类平面上，所以它们都满足：

$$w^T x' + b = 0$$

$$w^T x'' + b = 0$$

同时可以得到：$w^T x' = -b$，$w^T x'' = -b$，则有：

$$w^T(x'' - x') = w^T x'' - w^T x' = -b - (-b) = 0$$

(x"-x')是平面上的任一向量，(x"-x')与w内积为0，表示(x"-x')垂直于w，那么w就是平面的法向量。

现在，若要计算平面外一点x到该平面的距离，做法是只要将向量(x-x')投影到垂直于该平面的方向（即w方向）上就可以了。那么，令(x"-x')与w的夹角为θ，距离就可以表示为：



$$distance(x,b,w) = |(x-x')cos(\theta)| = |\ ||x-x'|| \cdot \frac{(x-x')w}{||x-x'|| \cdot ||w||}| = \frac{1}{||w||}|w^Tx - w^Tx'|$$

代入$w^Tx' = -b$，可得：

$$distance(x,b,w) = \frac{1}{||w||}|w^Tx + b|$$

点到分类面（Separating Hyperplane）的距离已经算出来了。基于这个分类面，所有的点均满足：$y_n(w^Tx_n + b) > 0$，表示所有点都分类正确，则distance公式就可以变换成：

$$distance(x,b,w) = \frac{1}{||w||}y_n(w^Tx_n + b)$$

那么，我们的目标形式就转换为：

$$\begin{aligned} \max_{b,\mathbf{w}} \quad & \text{margin}(b,\mathbf{w}) \\ \text{subject to} \quad & \text{every } y_n(\mathbf{w}^T\mathbf{x}_n + b) > 0 \\ & \text{margin}(b,\mathbf{w}) = \min_{n=1,\dots,N} \frac{1}{\|\mathbf{w}\|} y_n(\mathbf{w}^T\mathbf{x}_n + b) \end{aligned}$$

对上面的式子还不容易求解，我们继续对它进行简化。我们知道分类面$w^Tx + b = 0$和$3w^Tx + 3b = 0$其实是一样的。也就是说，对w和b进行同样的缩放还会得到同一分类面。所以，为了简化计算，我们令距离分类满最近的点满足$y_n(w^Tx_n + b) = 1$。那我们所要求的margin就变成了：

$$margin(b,w) = \frac{1}{||w||}$$

这样，目标形式就简化为：

$$\begin{aligned} \max_{b,\mathbf{w}} \quad & \frac{1}{\|\mathbf{w}\|} \\ \text{subject to} \quad & \text{every } y_n(\mathbf{w}^T\mathbf{x}_n + b) > 0 \\ & \min_{n=1,\dots,N} y_n(\mathbf{w}^T\mathbf{x}_n + b) = 1 \end{aligned}$$

这里可以省略条件：$y_n(w^Tx_n + b) > 0$，因为满足条件$y_n(w^Tx_n + b) = 1$必然满足大于零的条件。我们的目标就是根据这个条件，计算$\frac{1}{||w||}$的最大值。

刚刚我们讲的距离分类满最近的点满足$y_n(w^Tx_n + b) = 1$，也就是说对所有的点满足$y_n(w^Tx_n + b) \geq 1$。另外，因为最小化问题我们最熟悉也最好解，所以可以把目标$\frac{1}{||w||}$最大化转化为计算$\frac{1}{2}w^Tw$的最小化问题。



necessary constraints: $y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1$ for all n

original constraint: $\min_{n=1,\ldots,N} \; y_n(\mathbf{w}^T\mathbf{x}_n + b) = 1$
want: optimal $(b, \mathbf{w})$ **here (inside)**

if optimal $(b, \mathbf{w})$ outside, e.g. $y_n(\mathbf{w}^T\mathbf{x}_n + b) > 1.126$ for all n
—can scale $(b, \mathbf{w})$ to "more optimal" $\left(\frac{b}{1.126}, \frac{\mathbf{w}}{1.126}\right)$ **(contradiction!)**

final change: max $\Longrightarrow$ min, remove $\sqrt{\ }$, add $\frac{1}{2}$

$$\min_{b,\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$
$$\text{subject to} \quad y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \text{ for all } n$$

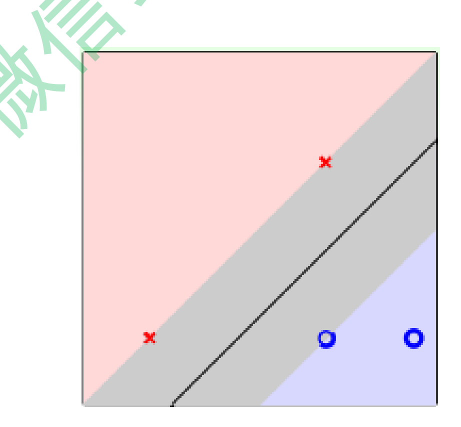
如上图所示，最终的条件就是$y_n(w^Tx_n + b) \geq 1$，而我们的目标就是最小化$\frac{1}{2}w^Tw$值。

Support Vector Machine

现在，条件和目标变成：

$$\min_{b,\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$
$$\text{subject to} \quad y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1 \text{ for all } n$$

现在，举个例子，假如平面上有四个点，两个正类，两个负类：



不同点的坐标加上条件$y_n(w^Tx_n + b) \geq 1$，可以得到：



$$\mathbf{X}=\begin{bmatrix}0&0\\2&2\\2&0\\3&0\end{bmatrix}\quad \mathbf{y}=\begin{bmatrix}-1\\-1\\+1\\+1\end{bmatrix}\quad \begin{array}{rrrl}&&-b\geq 1&(i)\\-2w_1&-2w_2&-b\geq 1&(ii)\\2w_1&&+b\geq 1&(iii)\\3w_1&&+b\geq 1&(iv)\end{array}$$

- $\left\{\begin{array}{lllll}(i)&\&&(iii)&\Longrightarrow&w_1\geq +1\\(ii)&\&&(iii)&\Longrightarrow&w_2\leq -1\end{array}\right\}\Longrightarrow \frac{1}{2}\mathbf{w}^T\mathbf{w}\geq 1$
- $(w_1=1, w_2=-1, b=-1)$ at **lower bound** and satisfies $(i)-(iv)$

最终，我们得到的条件是：

$$w_1\geq +1$$

$$w_2\leq -1$$

而我们的目标是：

$$min\ \frac{1}{2}w^Tw=\frac{1}{2}(w_1^2+w_2^2)\geq 1$$

目标最小值为1，即$w_1=1$，$w_2=-1$，$b=-1$，那么这个例子就得到了最佳分类面的解，如图所示，且$margin(b,w)=\frac{1}{||w||}=\frac{1}{\sqrt{2}}$。分类面的表达式为：

$$x_1-x_2-1=0$$

最终我们得到的矩的表达式为：

$$g_{SVM}(x)=sign(x_1-x_2-1)$$

Support Vector Machine(SVM)这个名字从何而来？为什么把这种分类面解法称为支持向量机呢？这是因为分类面仅仅由分类面的两边距离它最近的几个点决定的，其它点对分类面没有影响。决定分类面的几个点称之为支持向量（Support Vector），好比这些点“支撑”着分类面。而利用Support Vector得到最佳分类面的方法，称之为支持向量机（Support Vector Machine）。

下面介绍SVM的一般求解方法。先写下我们的条件和目标：

$$\begin{array}{ll}\min\limits_{b,\mathbf{w}} & \frac{1}{2}\mathbf{w}^T\mathbf{w}\\ \text{subject to} & y_n(\mathbf{w}^T\mathbf{x}_n+b)\geq 1 \text{ for all } n\end{array}$$

这是一个典型的二次规划问题，即Quadratic Programming（QP）。因为SVM的目标是关于w的二次函数，条件是关于w和b的一次函数，所以，它的求解过程还是比较容易的，可以使用一些软件（例如Matlab）自带的二次规划的库函数来求解。下图给出SVM与标准二次规划问题的参数对应关系：



optimal $(b, \mathbf{w}) = ?$

$$\min_{b,\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{subject to} \quad y_n(\mathbf{w}^T\mathbf{x}_n + b) \geq 1, \quad \text{for } n = 1, 2, \ldots, N$$

optimal $\mathbf{u} \leftarrow \text{QP}(\mathrm{Q}, \mathbf{p}, \mathrm{A}, \mathbf{c})$

$$\min_{\mathbf{u}} \quad \frac{1}{2}\mathbf{u}^T\mathrm{Q}\mathbf{u} + \mathbf{p}^T\mathbf{u}$$

$$\text{subject to} \quad \mathbf{a}_m^T\mathbf{u} \geq c_m, \quad \text{for } m = 1, 2, \ldots, M$$

objective function: $\mathbf{u} = \begin{bmatrix} b \\ \mathbf{w} \end{bmatrix}; \mathrm{Q} = \begin{bmatrix} 0 & \mathbf{0}_d^T \\ \mathbf{0}_d & \mathrm{I}_d \end{bmatrix}; \mathbf{p} = \mathbf{0}_{d+1}$

constraints: $\mathbf{a}_n^T = y_n \begin{bmatrix} 1 & \mathbf{x}_n^T \end{bmatrix}; c_n = 1; M = N$

那么，线性SVM算法可以总结为三步：

- **计算对应的二次规划参数Q，p，A，c**
- **根据二次规划库函数，计算b，w**
- **将b和w代入g_{SVM}，得到最佳分类面**

Linear Hard-Margin SVM Algorithm

1. $\mathrm{Q} = \begin{bmatrix} 0 & \mathbf{0}_d^T \\ \mathbf{0}_d & \mathrm{I}_d \end{bmatrix}; \mathbf{p} = \mathbf{0}_{d+1}; \mathbf{a}_n^T = y_n \begin{bmatrix} 1 & \mathbf{x}_n^T \end{bmatrix}; c_n = 1$
2. $\begin{bmatrix} b \\ \mathbf{w} \end{bmatrix} \leftarrow \text{QP}(\mathrm{Q}, \mathbf{p}, \mathrm{A}, \mathbf{c})$
3. return b & $\mathbf{w}$ as g_{SVM}

这种方法称为Linear Hard-Margin SVM Algorithm。如果是非线性的，例如包含x的高阶项，那么可以使用我们之前在《机器学习基石》课程中介绍的特征转换的方法，先作$z_n = \Phi(x_n)$的特征变换，从非线性的x域映射到线性的z域空间，再利用Linear Hard-Margin SVM Algorithm求解即可。

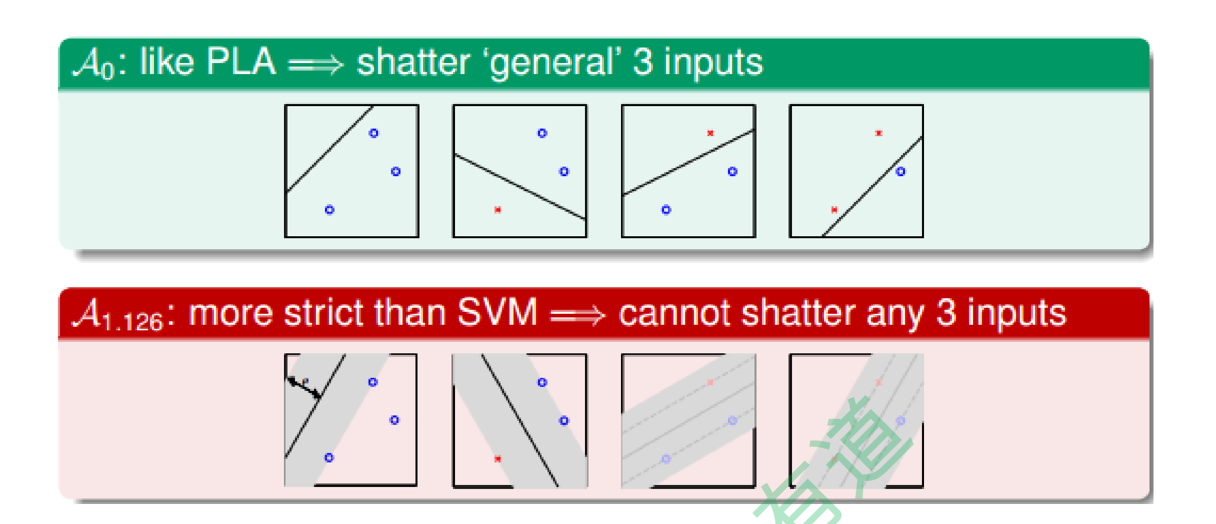
Reasons behind Large-Margin Hyperplane

从视觉和直觉的角度，我们认为Large-Margin Hyperplane的分类效果更好。SVM的这种思想其实与我们之前介绍的机器学习非常重要的正则化regularization思想很类似。regularization的目标是将E_{in}最小化，条件是$w^Tw \leq C$；SVM的目标是w^Tw最小化，条件是$y_n(w^Tx_n + b) \geq 1$，即保证了$E_{in} = 0$。有趣的是，regularization与SVM的目标和限制条件分别对调了。其实，考虑的内容是类似的，效果也是相近的。SVM也可以说是一种weight-decay regularization，限制条件是$E_{in} = 0$。



	minimize	constraint
regularization	E_{in}	$\mathbf{w}^T\mathbf{w} \le C$
SVM	$\mathbf{w}^T\mathbf{w}$	$E_{in} = 0$ [and more]

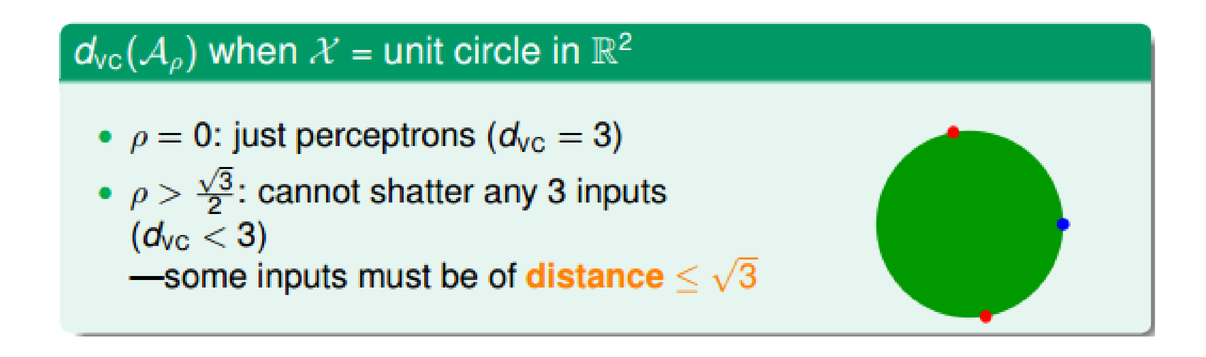
从另一方面来看，Large-Margin会限制Dichotomies的个数。这从视觉上也很好理解，假如一条分类面越“胖”，即对应Large-Margin，那么它可能shtter的点的个数就可能越少：



之前的《机器学习基石》课程中介绍过，Dichotomies与VC Dimension是紧密联系的。也就是说如果Dichotomies越少，那么复杂度就越低，即有效的VC Dimension就越小，得到$E_{out} \approx E_{in}$，泛化能力强。

下面我们从概念的角度推导一下为什么dichotomies越少，VC Dimension就越少。首先我们考虑一下Large-Margin演算法的VC Dimension，记为$d_{vc}(A_\rho)$。$d_{vc}(A_\rho)$与数据有关，而我们之前介绍的d_{vc}与数据无关。

假如平面上有3个点分布在单位圆上，如果Margin为0，即$\rho = 0$，这条细细的直线可以很容易将圆上任意三点分开（shatter），就能得到它的$d_{vc} = 3$。如果$\rho > \frac{\sqrt{3}}{2}$，这条粗粗的线无论如何都不能将圆上的任一三点全完分开（no shatter），因为圆上必然至少存在两个点的距离小于$\sqrt{3}$，那么其对应d的$d_{vc} < 3$。



那么，一般地，在d维空间，当数据点分布在半径为R的超球体内时，得到的$d_{vc}(A_\rho)$满足下列不等式：



generally, when $\mathcal{X}$ in radius-R hyperball:

$$d_{\mathrm{VC}}(\mathcal{A}_\rho) \le \min\left(\frac{R^2}{\rho^2}, d\right) + 1 \le \underbrace{d+1}_{d_{\mathrm{VC}}(\text{perceptrons})}$$

之前介绍的Perceptrons的VC Dimension为d+1，这里得到的结果是Large-Margin演算法的 $d_{vc}(A_\rho) \le d+1$。所以，由于Large-Margin，得到的dichotomies个数减少，从而VC Dimension也减少了。VC Dimension减少降低了模型复杂度，提高了泛化能力。

总结

本节课主要介绍了线性支持向量机（Linear Support Vector Machine）。我们先从视觉角度出发，希望得到一个比较“胖”的分类面，即满足所有的点距离分类面都尽可能远。然后，我们通过一步步推导和简化，最终把这个问题转换为标准的二次规划（QP）问题。二次规划问题可以使用Matlab等软件来进行求解，得到我们要求的w和b，确定分类面。这种方法背后的原理其实就是减少了dichotomies的种类，减少了有效的VC Dimension数量，从而让机器学习的模型具有更好的泛化能力。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程



林轩田《机器学习技法》课程笔记2 -- Dual Support Vector Machine

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了线性支持向量机（Linear Support Vector Machine）。Linear SVM的目标是找出最“胖”的分割线进行正负类的分离，方法是使用二次规划来求出分类线。本节课将从另一个方面入手，研究对偶支持向量机（Dual Support Vector Machine），尝试从新的角度计算得出分类线，推广SVM的应用范围。

Motivation of Dual SVM

首先，我们回顾一下，对于非线性SVM，我们通常可以使用非线性变换将变量从x域转换到z域中。然后，在z域中，根据上一节课的内容，使用线性SVM解决问题即可。上一节课我们说过，使用SVM得到large-margin，减少了有效的VC Dimension，限制了模型复杂度；另一方面，使用特征转换，目的是让模型更复杂，减小E_{in}。所以说，非线性SVM是把这两者目的结合起来，平衡这两者的关系。那么，特征转换下，求解QP问题在z域中的维度设为$\hat{d}+1$，如果模型越复杂，则$\hat{d}+1$越大，相应求解这个QP问题也变得很困难。当$\hat{d}$无限大的时候，问题将会变得难以求解，那么有没有什么办法可以解决这个问题呢？一种方法就是使SVM的求解过程不依赖$\hat{d}$，这就是我们本节课所要讨论的主要内容。

$$\min_{b,\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$
$$\text{s. t.} \quad y_n(\mathbf{w}^T \underbrace{\mathbf{z}_n}_{\Phi(\mathbf{x}_n)} + b) \geq 1, \quad \text{for } n = 1, 2, \ldots, N$$

Non-Linear Hard-Margin SVM

1. $\mathrm{Q} = \begin{bmatrix} 0 & \mathbf{0}_{\tilde{d}}^T \\ \mathbf{0}_{\tilde{d}} & \mathrm{I}_{\tilde{d}} \end{bmatrix}; \mathbf{p} = \mathbf{0}_{\tilde{d}+1};$ $\mathbf{a}_n^T = y_n \begin{bmatrix} 1 & \mathbf{z}_n^T \end{bmatrix}; c_n = 1$
2. $\begin{bmatrix} b \\ \mathbf{w} \end{bmatrix} \leftarrow \mathrm{QP}(\mathrm{Q}, \mathbf{p}, \mathrm{A}, \mathbf{c})$
3. return $b \in \mathbb{R}$ & $\mathbf{w} \in \mathbb{R}^{\tilde{d}}$ with $g_{\text{SVM}}(\mathbf{x}) = \text{sign}(\mathbf{w}^T\Phi(\mathbf{x}) + b)$

比较一下，我们上一节课所讲的Original SVM二次规划问题的变量个数是$\hat{d}+1$，有



个限制条件；而本节课，我们把问题转化为对偶问题（'Equivalent' SVM），同样是二次规划，只不过变量个数变成N个，有N+1个限制条件。这种对偶SVM的好处就是问题只跟N有关，与$\hat{d}$无关，这样就不存在上文提到的当$\hat{d}$无限大时难以求解的情况。

Original SVM

(convex) QP of
- $\tilde{d}+1$ variables
- N constraints

'Equivalent' SVM

(convex) QP of
- N variables
- $N+1$ constraints

如何把问题转化为对偶问题（'Equivalent' SVM），其中的数学推导非常复杂，本文不做详细数学论证，但是会从概念和原理上进行简单的推导。

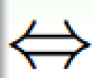
还记得我们在《机器学习基石》课程中介绍的Regularization中，在最小化E_{in}的过程中，也添加了限制条件：$w^Tw\leq C$。我们的求解方法是引入拉格朗日因子λ，将有条件的最小化问题转换为无条件的最小化问题：
$min\ E_{aug}(w)=E_{in}(w)+\frac{\lambda}{N}w^Tw$，最终得到的w的最优化解为：

$$\nabla E_{in}(w)+\frac{2\lambda}{N}w=0$$

所以，在regularization问题中，λ是已知常量，求解过程变得容易。那么，对于dual SVM问题，同样可以引入λ，将条件问题转换为非条件问题，只不过λ是未知参数，且个数是N，需要对其进行求解。

Regularization by Constrained-Minimizing E_{in}

$$\min_{\mathbf{w}} E_{in}(\mathbf{w}) \text{ s.t. } \mathbf{w}^T\mathbf{w}\leq C$$



Regularization by Minimizing E_{aug}

$$\min_{\mathbf{w}} E_{aug}(\mathbf{w}) = E_{in}(\mathbf{w})+\frac{\lambda}{N}\mathbf{w}^T\mathbf{w}$$

- C equivalent to some $\lambda\geq 0$ by checking **optimality condition**

$$\nabla E_{in}(\mathbf{w})+\frac{2\lambda}{N}\mathbf{w}=\mathbf{0}$$

- regularization: view λ as **given parameter instead of C**, and solve 'easily'
- dual SVM: view λ's as unknown given the constraints, and **solve them as variables instead**



如何将条件问题转换为非条件问题？上一节课我们介绍的SVM中，目标是：$min\ \frac{1}{2}w^Tw$，条件是：$y_n(w^Tz_n+b)\geq 1,\ for\ n=1,2,\cdots,N$。首先，我们令拉格朗日因子为$\alpha_n$（区别于regularization），构造一个函数：

$$L(b,w,\alpha)=\frac{1}{2}w^Tw+\sum_{n=1}^{N}\alpha_n(1-y_n(w^Tz_n+b))$$

这个函数右边第一项是SVM的目标，第二项是SVM的条件和拉格朗日因子α_n的乘积。我们把这个函数称为拉格朗日函数，其中包含三个参数：b，w，α_n。

$$\min_{b,w}\ \frac{1}{2}w^Tw \quad \text{s.t.}\ y_n(w^Tz_n+b)\geq 1,\ \text{for } n=1,2,\ldots,N$$

Lagrange Function

with Lagrange multipliers ~~λ_n~~ α_n,

$$\mathcal{L}(b,w,\alpha)=\underbrace{\frac{1}{2}w^Tw}_{\text{objective}}+\sum_{n=1}^{N}\alpha_n\underbrace{(1-y_n(w^Tz_n+b))}_{\text{constraint}}$$

下面，我们利用拉格朗日函数，把SVM构造成一个非条件问题：

Claim

$$\text{SVM}\equiv\min_{b,w}\left(\max_{\text{all }\alpha_n\geq 0}\mathcal{L}(b,w,\alpha)\right)=\min_{b,w}\left(\infty \text{ if violate};\ \frac{1}{2}w^Tw \text{ if feasible}\right)$$

- any 'violating' (b,w): $\max_{\text{all }\alpha_n\geq 0}\left(\square+\sum_n\alpha_n(\text{some positive})\right)\rightarrow\infty$
- any 'feasible' (b,w): $\max_{\text{all }\alpha_n\geq 0}\left(\square+\sum_n\alpha_n(\text{all non-positive})\right)=\square$

该最小化问题中包含了最大化问题，怎么解释呢？首先我们规定拉格朗日因子$\alpha_n\geq 0$，根据SVM的限定条件可得：$(1-y_n(w^Tz_n+b))\leq 0$，如果没有达到最优解，即有不满足$(1-y_n(w^Tz_n+b))\leq 0$的情况，因为$\alpha_n\geq 0$，那么必然有$\sum_n\alpha_n(1-y_n(w^Tz_n+b))\geq 0$。对于这种大于零的情况，其最大值是无解的。如果对于所有的点，均满足$(1-y_n(w^Tz_n+b))\leq 0$，那么必然有$\sum_n\alpha_n(1-y_n(w^Tz_n+b))\leq 0$，则当$\sum_n\alpha_n(1-y_n(w^Tz_n+b))=0$时，其有最大值，最大值就是我们SVM的目标：$\frac{1}{2}w^Tw$。因此，这种转化为非条件的SVM构造函数的形式是可行的。

Lagrange Dual SVM

现在，我们已经将SVM问题转化为与拉格朗日因子α_n有关的最大最小值形式。已知$\alpha_n \geq 0$，那么对于任何固定的α'，且$\alpha'_n \geq 0$，一定有如下不等式成立：

for any fixed α' with all $\alpha'_n \geq 0$,

$$\min_{b,\mathbf{w}} \left(\max_{\text{all } \alpha_n \geq 0} \mathcal{L}(b, \mathbf{w}, \alpha) \right) \geq \min_{b,\mathbf{w}} \mathcal{L}(b, \mathbf{w}, \alpha')$$

对上述不等式右边取最大值，不等式同样成立：

for best $\alpha' \geq \mathbf{0}$ on RHS,

$$\min_{b,\mathbf{w}} \left(\max_{\text{all } \alpha_n \geq 0} \mathcal{L}(b, \mathbf{w}, \alpha) \right) \geq \underbrace{\max_{\text{all } \alpha'_n \geq 0} \min_{b,\mathbf{w}} \mathcal{L}(b, \mathbf{w}, \alpha')}_{\text{Lagrange dual problem}}$$

上述不等式表明，我们对SVM的min和max做了对调，满足这样的关系，这叫做Lagrange dual problem。不等式右边是SVM问题的下界，我们接下来的目的就是求出这个下界。

已知≥是一种弱对偶关系，在二次规划QP问题中，如果满足以下三个条件：

- **函数是凸的（convex primal）**
- **函数有解（feasible primal）**
- **条件是线性的（linear constraints）**

那么，上述不等式关系就变成强对偶关系，≥变成=，即一定存在满足条件的解(b, w, α)，使等式左边和右边都成立，SVM的解就转化为右边的形式。

经过推导，SVM对偶问题的解已经转化为无条件形式：



$$\max_{\text{all } \alpha_n \ge 0} \left(\underbrace{\min_{b,\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{n=1}^{N} \alpha_n \left(1 - y_n(\mathbf{w}^T\mathbf{z}_n + b)\right)}_{\mathcal{L}(b,\mathbf{w},\alpha)} \right)$$

其中，上式括号里面的是对拉格朗日函数$L(b,w,\alpha)$计算最小值。那么根据梯度下降算法思想：最小值位置满足梯度为零。首先，令$L(b,w,\alpha)$对参数b的梯度为零：

$$\frac{\partial L(b,w,\alpha)}{\partial b} = 0 = -\sum_{n=1}^{N} \alpha_n y_n$$

也就是说，最优解一定满足$\sum_{n=1}^{N} \alpha_n y_n = 0$。那么，我们把这个条件代入计算max条件中（与$\alpha_n \ge 0$同为条件），并进行化简：

$$\max_{\text{all } \alpha_n \ge 0, \sum y_n \alpha_n = 0} \left(\min_{b,\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{n=1}^{N} \alpha_n \left(1 - y_n(\mathbf{w}^T\mathbf{z}_n)\right) \right)$$

这样，SVM表达式消去了b，问题化简了一些。然后，再根据最小值思想，令$L(b,w,\alpha)$对参数w的梯度为零：

$$\frac{\partial L(b,w,\alpha}{\partial w} = 0 = w - \sum_{n=1}^{N} \alpha_n y_n z_n$$

即得到：

$$w = \sum_{n=1}^{N} \alpha_n y_n z_n$$

也就是说，最优解一定满足$w = \sum_{n=1}^{N} \alpha_n y_n z_n$。那么，同样我们把这个条件代入并进行化简：



$$\max_{\text{all } \alpha_n \geq 0, \sum y_n \alpha_n = 0, \mathbf{w} = \sum \alpha_n y_n \mathbf{z}_n} \left(\min_{b, \mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{n=1}^{N} \alpha_n - \mathbf{w}^T\mathbf{w} \right)$$

$$\Longleftrightarrow \max_{\text{all } \alpha_n \geq 0, \sum y_n \alpha_n = 0, \mathbf{w} = \sum \alpha_n y_n \mathbf{z}_n} -\frac{1}{2}\|\sum_{n=1}^{N} \alpha_n y_n \mathbf{z}_n\|^2 + \sum_{n=1}^{N} \alpha_n$$

这样，SVM表达式消去了w，问题更加简化了。这时候的条件有3个：

- all $\alpha_n \geq 0$
- $\sum_{n=1}^N \alpha_n y_n = 0$
- $w = \sum_{n=1}^N \alpha_n y_n z_n$

SVM简化为只有α_n的最佳化问题，即计算满足上述三个条件下，函数$-\frac{1}{2}||\sum_{n=1}^N \alpha_n y_n z_n||^2 + \sum_{n=1}^N \alpha_n$最小值时对应的$\alpha_n$是多少。

总结一下，SVM最佳化形式转化为只与α_n有关：

$$\max_{\text{all } \alpha_n \geq 0, \sum y_n \alpha_n = 0, \mathbf{w} = \sum \alpha_n y_n \mathbf{z}_n} -\frac{1}{2}\|\sum_{n=1}^{N} \alpha_n y_n \mathbf{z}_n\|^2 + \sum_{n=1}^{N} \alpha_n$$

其中，满足最佳化的条件称之为Karush-Kuhn-Tucker(KKT)：

if primal-dual optimal $(b, \mathbf{w}, \alpha)$,

- primal feasible: $y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1$
- dual feasible: $\alpha_n \geq 0$
- dual-inner optimal: $\sum y_n \alpha_n = 0$; $\mathbf{w} = \sum \alpha_n y_n \mathbf{z}_n$
- primal-inner optimal (at optimal all 'Lagrange terms' disappear):

$$\alpha_n(1 - y_n(\mathbf{w}^T\mathbf{z}_n + b)) = 0$$

—called **Karush-Kuhn-Tucker (KKT) conditions**, necessary for optimality [& sufficient here]

在下一部分中，我们将利用KKT条件来计算最优化问题中的α，进而得到b和w。



Solving Dual SVM

上面我们已经得到了dual SVM的简化版了，接下来，我们继续对它进行一些优化。首先，将max问题转化为min问题，再做一些条件整理和推导，得到：

standard hard-margin SVM dual

$$\min_{\alpha} \quad \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}\alpha_n\alpha_m y_n y_m \mathbf{z}_n^T\mathbf{z}_m - \sum_{n=1}^{N}\alpha_n$$

$$\text{subject to} \quad \sum_{n=1}^{N} y_n\alpha_n = 0;$$

$$\alpha_n \ge 0, \text{for } n = 1, 2, \ldots, N$$

(convex) QP of N variables & $N+1$ constraints, as promised

显然，这是一个convex的QP问题，且有N个变量α_n，限制条件有N+1个。则根据上一节课讲的QP解法，找到Q，p，A，c对应的值，用软件工具包进行求解即可。

optimal $\alpha = ?$

$$\min_{\alpha} \quad \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}\alpha_n\alpha_m y_n y_m \mathbf{z}_n^T\mathbf{z}_m - \sum_{n=1}^{N}\alpha_n$$

$$\text{subject to} \quad \sum_{n=1}^{N} y_n\alpha_n = 0;$$

$$\alpha_n \ge 0, \text{ for } n = 1, 2, \ldots, N$$

optimal $\alpha \leftarrow \mathrm{QP}(\mathrm{Q}, \mathbf{p}, \mathrm{A}, \mathbf{c})$

$$\min_{\alpha} \quad \frac{1}{2}\alpha^T \mathrm{Q}\alpha + \mathbf{p}^T\alpha$$

$$\text{subject to} \quad \mathbf{a}_i^T\alpha \ge c_i, \text{ for } i = 1, 2, \ldots$$

- $q_{n,m} = y_n y_m \mathbf{z}_n^T\mathbf{z}_m$
- $\mathbf{p} = -\mathbf{1}_N$
- $\mathbf{a}_{\ge} = \mathbf{y}$, $\mathbf{a}_{\le} = -\mathbf{y}$; $\mathbf{a}_n^T = n$-th unit direction
- $c_{\ge} = 0$, $c_{\le} = 0$; $c_n = 0$

求解过程很清晰，但是值得注意的是，$q_{n,m} = y_n y_m z_n^T z_m$，大部分值是非零的，称为dense。当N很大的时候，例如N=30000，那么对应的Q_D的计算量将会很大，存储空间也很大。所以一般情况下，对dual SVM问题的矩阵Q_D，需要使用一些特殊的方法，这部分内容就不再赘述了。

- $q_{n,m} = y_n y_m \mathbf{z}_n^T \mathbf{z}_m$, often non-zero
- if $N = 30,000$, dense Q_D (N by N symmetric) takes > 3G RAM
- need **special solver** for
  - not storing whole Q_D
  - utilizing special constraints properly

  to scale up to large N

得到α_n之后，再根据之前的KKT条件，就可以计算出w和b了。首先利用条件$w = \sum \alpha_n y_n z_n$得到w，然后利用条件$\alpha_n(1 - y_n(w^T z_n + b)) = 0$，取任一$\alpha_n \neq 0$即$\alpha_n$>0的点，得到$1 - y_n(w^T z_n + b) = 0$，进而求得$b = y_n - w^T z_n$。

KKT conditions

if primal-dual optimal $(b, \mathbf{w}, \alpha)$,

- primal feasible: $y_n(\mathbf{w}^T \mathbf{z}_n + b) \geq 1$
- dual feasible: $\alpha_n \geq 0$
- dual-inner optimal: $\sum y_n \alpha_n = 0$; $\mathbf{w} = \sum \alpha_n y_n \mathbf{z}_n$
- primal-inner optimal (at optimal all 'Lagrange terms' disappear):

$$\alpha_n(1 - y_n(\mathbf{w}^T \mathbf{z}_n + b)) = 0 \text{ (complementary slackness)}$$

- optimal $\alpha \Longrightarrow$ optimal $\mathbf{w}$? easy above!
- optimal $\alpha \Longrightarrow$ optimal b? a range from primal feasible & equality from **comp. slackness** if one $\alpha_n > 0 \Rightarrow b = y_n - \mathbf{w}^T \mathbf{z}_n$

值得注意的是，计算b值，α_n>0时，有$y_n(w^T z_n + b) = 1$成立。$y_n(w^T z_n + b) = 1$正好表示的是该点在SVM分类线上，即fat boundary。也就是说，满足α_n>0的点一定落在fat boundary上，这些点就是Support Vector。这是一个非常有趣的特性。

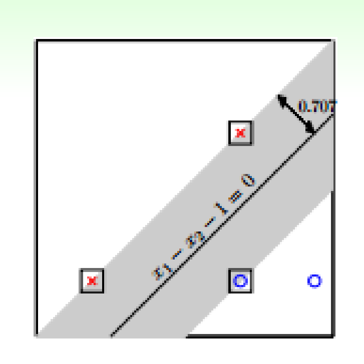
Messages behind Dual SVM

回忆一下，上一节课中，我们把位于分类线边界上的点称为support vector（candidates）。本节课前面介绍了α_n>0的点一定落在分类线边界上，这些点称之为support vector（注意没有candidates）。也就是说分类线上的点不一定都是支



持向量，但是满足α_n>0的点，一定是支持向量。

- on boundary: 'locates' fattest hyperplane; others: **not needed**
- examples with $\alpha_n > 0$: on boundary
- call $\alpha_n > 0$ examples $(\mathbf{z}_n, y_n)$ **support vectors** ~~(candidates)~~
- SV (positive α_n) $\subseteq$ SV candidates (on boundary)



SV只由α_n>0的点决定，根据上一部分推导的w和b的计算公式，我们发现，w和b仅由SV即α_n>0的点决定，简化了计算量。这跟我们上一节课介绍的分类线只由“胖”边界上的点所决定是一个道理。也就是说，样本点可以分成两类：一类是support vectors，通过support vectors可以求得fattest hyperplane；另一类不是support vectors，对我们求得fattest hyperplane没有影响。

- only SV needed to compute $\mathbf{w}$: $\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{z}_n = \sum_{\text{SV}} \alpha_n y_n \mathbf{z}_n$
- only SV needed to compute b: $b = y_n - \mathbf{w}^T \mathbf{z}_n$ with any SV $(\mathbf{z}_n, y_n)$

回过头来，我们来比较一下SVM和PLA的w公式：

SVM

$$\mathbf{w}_{\text{SVM}} = \sum_{n=1}^{N} \alpha_n (y_n \mathbf{z}_n)$$

α_n from **dual solution**

PLA

$$\mathbf{w}_{\text{PLA}} = \sum_{n=1}^{N} \beta_n (y_n \mathbf{z}_n)$$

β_n by **# mistake corrections**

我们发现，二者在形式上是相似的。w_{SVM}由fattest hyperplane边界上所有的SV决定，w_{PLA}由所有当前分类错误的点决定。w_{SVM}和w_{PLA}都是原始数据点$y_n z_n$的线性组合形式，是原始数据的代表。



$$\mathbf{w} = \text{linear combination of } y_n \mathbf{z}_n$$

- also true for GD/SGD-based LogReg/LinReg when $\mathbf{w}_0 = \mathbf{0}$
- call $\mathbf{w}$ **'represented' by data**

总结一下，本节课和上节课主要介绍了两种形式的SVM，一种是Primal Hard-Margin SVM，另一种是Dual Hard_Margin SVM。Primal Hard-Margin SVM有$\hat{d}+1$个参数，有N个限制条件。当$\hat{d}+1$很大时，求解困难。而Dual Hard_Margin SVM有N个参数，有N+1个限制条件。当数据量N很大时，也同样会增大计算难度。两种形式都能得到w和b，求得fattest hyperplane。通常情况下，如果N不是很大，一般使用Dual SVM来解决问题。

Primal Hard-Margin SVM	Dual Hard-Margin SVM
$\min_{b,\mathbf{w}} \ \frac{1}{2}\mathbf{w}^T\mathbf{w}$	$\min_{\alpha} \ \frac{1}{2}\alpha^T \mathrm{Q}_\mathrm{D}\alpha - \mathbf{1}^T\alpha$
sub. to $y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1$, for $n = 1, 2, \ldots, N$	s.t. $\mathbf{y}^T\alpha = 0$; $\alpha_n \geq 0$ for $n = 1, \ldots, N$
• $\tilde{d}+1$ variables, N constraints —suitable when $\tilde{d}+1$ small	• N variables, $N+1$ simple constraints —suitable when N small
• physical meaning: locate specially-scaled $(b, \mathbf{w})$	• physical meaning: locate SVs $(\mathbf{z}_n, y_n)$ & their α_n

both eventually result in optimal $(b, \mathbf{w})$ for fattest hyperplane

$$g_{\mathrm{SVM}}(\mathbf{x}) = \mathrm{sign}(\mathbf{w}^T \mathbf{\Phi}(\mathbf{x}) + b)$$

这节课提出的Dual SVM的目的是为了避免计算过程中对$\hat{d}$的依赖，而只与N有关。但是，Dual SVM是否真的消除了对$\hat{d}$的依赖呢？其实并没有。因为在计算$q_{n,m} = y_n y_m z_n^T z_m$的过程中，由z向量引入了$\hat{d}$，实际上复杂度已经隐藏在计算过程中了。所以，我们的目标并没有实现。下一节课我们将继续研究探讨如何消除对$\hat{d}$的依赖。



$$\min_{\alpha} \quad \frac{1}{2}\alpha^T Q_D \alpha - \mathbf{1}^T\alpha$$
$$\text{subject to} \quad \mathbf{y}^T\alpha = 0;$$
$$\alpha_n \geq 0, \text{ for } n = 1, 2, \ldots, N$$

- N variables, $N+1$ constraints: **no dependence on $\tilde{d}$?**
- $q_{n,m} = y_n y_m \mathbf{z}_n^T \mathbf{z}_m$: inner product in $\mathbb{R}^{\tilde{d}}$ —$O(\tilde{d})$ via naïve computation!

总结

本节课主要介绍了SVM的另一种形式：Dual SVM。我们这样做的出发点是为了移除计算过程对$\hat{d}$的依赖。Dual SVM的推导过程是通过引入拉格朗日因子α，将SVM转化为新的非条件形式。然后，利用QP，得到最佳解的拉格朗日因子α。再通过KKT条件，计算得到对应的w和b。最终求得fattest hyperplane。下一节课，我们将解决Dual SVM计算过程中对$\hat{d}$的依赖问题。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程



林轩田《机器学习技法》课程笔记3 -- Kernel Support Vector Machine

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了SVM的对偶形式，即dual SVM。Dual SVM也是一个二次规划问题，可以用QP来进行求解。之所以要推导SVM的对偶形式是因为：首先，它展示了SVM的几何意义；然后，从计算上，求解过程“好像”与所在维度$\hat{d}$无关，规避了$\hat{d}$很大时难以求解的情况。但是，上节课的最后，我们也提到dual SVM的计算过程其实跟$\hat{d}$还是有关系的。那么，能不能完全摆脱对$\hat{d}$的依赖，从而减少SVM计算量呢？这就是我们本节课所要讲的主要内容。

Kernel Trick

我们上节课推导的dual SVM是如下形式：

half-way done:

$$\min_{\alpha} \quad \frac{1}{2}\alpha^T Q_D \alpha - \mathbf{1}^T\alpha$$

$$\text{subject to} \quad \mathbf{y}^T\alpha = 0;$$

$$\alpha_n \geq 0, \text{for } n = 1, 2, \ldots, N$$

- $q_{n,m} = y_n y_m \mathbf{z}_n^T \mathbf{z}_m$: inner product in $\mathbb{R}^{\tilde{d}}$
- need: $\mathbf{z}_n^T \mathbf{z}_m = \Phi(\mathbf{x}_n)^T \Phi(\mathbf{x}_m)$ calculated faster than $O(\tilde{d})$

其中α是拉格朗日因子，共N个，这是我们要求解的，而条件共有N+1个。我们来看向量Q_D中的$q_{n,m} = y_n y_m z_n^T z_m$，看似这个计算与$\hat{d}$无关，但是$z_n^T z_m$的内积中不得不引入$\hat{d}$。也就是说，如果$\hat{d}$很大，计算$z_n^T z_m$的复杂度也会很高，同样会影响QP问题的计算效率。可以说，$q_{n,m} = y_n y_m z_n^T z_m$这一步是计算的瓶颈所在。

其实问题的关键在于$z_n^T z_m$内积求解上。我们知道，z是由x经过特征转换而来：

$$z_n^T z_m = \Phi(x_n)\Phi(x_m)$$

如果从x空间来看的话，$z_n^T z_m$分为两个步骤：1. 进行特征转换$\Phi(x_n)$和$\Phi(x_m)$；2. 计算$\Phi(x_n)$与$\Phi(x_m)$的内积。这种先转换再计算内积的方式，必然会引入$\hat{d}$参数，从而在$\hat{d}$很大的时候影响计算速度。那么，若把这两个步骤联合起来，是否可以有效地减小计算量，提高计算速度呢？

我们先来看一个简单的例子，对于二阶多项式转换，各种排列组合为：



2nd order polynomial transform

$$\Phi_2(\mathbf{x}) = (1, x_1, x_2, \ldots, x_d, x_1^2, x_1x_2, \ldots, x_1x_d, x_2x_1, x_2^2, \ldots, x_2x_d, \ldots, x_d^2)$$

—include both x_1x_2 & x_2x_1 for '**simplicity**' :-)

这里提一下，为了简单起见，我们把$x_0 = 1$包含进来，同时将二次项x_1x_2和x_2x_1也包含进来。转换之后再做内积并进行推导，得到：

$$\begin{aligned}\Phi_2(\mathbf{x})^T\Phi_2(\mathbf{x}') &= 1 + \sum_{i=1}^{d} x_i x_i' + \sum_{i=1}^{d}\sum_{j=1}^{d} x_i x_j x_i' x_j' \\ &= 1 + \sum_{i=1}^{d} x_i x_i' + \sum_{i=1}^{d} x_i x_i' \sum_{j=1}^{d} x_j x_j' \\ &= 1 + \mathbf{x}^T\mathbf{x}' + (\mathbf{x}^T\mathbf{x}')(\mathbf{x}^T\mathbf{x}')\end{aligned}$$

其中x^Tx'是x空间中特征向量的内积。所以，$\Phi_2(x)$与$\Phi_2(x')$的内积的复杂度由原来的$O(d^2)$变成$O(d)$，只与x空间的维度d有关，而与z空间的维度$\hat{d}$无关，这正是我们想要的！

至此，我们发现如果把特征转换和z空间计算内积这两个步骤合并起来，有可能会简化计算。因为我们只是推导了二阶多项式会提高运算速度，这个特例并不具有一般推论性。但是，我们还是看到了希望。

我们把合并特征转换和计算内积这两个步骤的操作叫做Kernel Function，用大写字母K表示。例如刚刚讲的二阶多项式例子，它的kernel function为：

$$K_\Phi(x, x') = \Phi(x)^T\Phi(x')$$

$$K_{\Phi_2}(x, x') = 1 + (x^Tx') + (x^Tx')^2$$

有了kernel function之后，我们来看看它在SVM里面如何使用。在dual SVM中，二次项系数$q_{n,m}$中有z的内积计算，就可以用kernel function替换：

$$q_{n,m} = y_n y_m z_n^T z_m = y_n y_m K(x_n, x_m)$$

所以，直接计算出$K(x_n, x_m)$，再代入上式，就能得到$q_{n,m}$的值。

$q_{n,m}$值计算之后，就能通过QP得到拉格朗日因子α_n。然后，下一步就是计算b（取α_n>0的点，即SV），b的表达式中包含z，可以作如下推导：

$$b = y_s - w^T z_s = y_s - (\sum_{n=1}^{N} \alpha_n y_n z_n)^T z_s = y_s - \sum_{n=1}^{N} \alpha_n y_n (K(x_n, x_s))$$

这样得到的b就可以用kernel function表示，而与z空间无关。

最终我们要求的矩g_{SVM}可以作如下推导：



$$g_{SVM}(x)=sign(w^T\Phi(x)+b)=sign((\sum_{n=1}^{N}\alpha_ny_nz_n)^Tz+b)=sign(\sum_{n=1}^{N}\alpha_ny_n(K(x_n,x))+b)$$

至此，dual SVM中我们所有需要求解的参数都已经得到了，而且整个计算过程中都没有在z空间作内积，即与z无关。我们把这个过程称为kernel trick，也就是把特征转换和计算内积两个步骤结合起来，用kernel function来避免计算过程中受$\hat{d}$的影响，从而提高运算速度。

- quadratic coefficient $q_{n,m}=y_ny_m\mathbf{z}_n^T\mathbf{z}_m=y_ny_mK(\mathbf{x}_n,\mathbf{x}_m)$
- optimal bias b? from SV $(\mathbf{x}_s,y_s)$,

$$b=y_s-\mathbf{w}^T\mathbf{z}_s=y_s-\left(\sum_{n=1}^{N}\alpha_ny_n\mathbf{z}_n\right)^T\mathbf{z}_s=y_s-\sum_{n=1}^{N}\alpha_ny_n\left(K(\mathbf{x}_n,\mathbf{x}_s)\right)$$

- optimal hypothesis g_{SVM}: for test input $\mathbf{x}$,

$$g_{\text{SVM}}(\mathbf{x})=\text{sign}\left(\mathbf{w}^T\Phi(\mathbf{x})+b\right)=\text{sign}\left(\sum_{n=1}^{N}\alpha_ny_nK(\mathbf{x}_n,\mathbf{x})+b\right)$$

那么总结一下，引入kernel funtion后，SVM算法变成：

Kernel Hard-Margin SVM Algorithm

1. $q_{n,m}=y_ny_mK(\mathbf{x}_n,\mathbf{x}_m)$; $\mathbf{p}=-\mathbf{1}_N$; $(\mathrm{A},\mathbf{c})$ for equ./bound constraints
2. $\boldsymbol{\alpha}\leftarrow\text{QP}(\mathrm{Q_D},\mathbf{p},\mathrm{A},\mathbf{c})$
3. $b\leftarrow\left(y_s-\sum_{\text{SV indices } n}\alpha_ny_nK(\mathbf{x}_n,\mathbf{x}_s)\right)$ with SV $(\mathbf{x}_s,y_s)$
4. return SVs and their α_n as well as b such that for new $\mathbf{x}$,

$$g_{\text{SVM}}(\mathbf{x})=\text{sign}\left(\sum_{\text{SV indices } n}\alpha_ny_nK(\mathbf{x}_n,\mathbf{x})+b\right)$$

分析每个步骤的时间复杂度为：

- (1): time complexity $O(N^2)\cdot$ (kernel evaluation)
- (2): QP with N variables and $N+1$ constraints
- (3) & (4): time complexity $O(\#\text{SV})\cdot$ (kernel evaluation)

我们把这种引入kernel function的SVM称为kernel SVM，它是基于dual SVM推导而来的。kernel SVM同样只用SV（α_n>0）就能得到最佳分类面，而且整个计算过程中摆脱了$\hat{d}$的影响，大大提高了计算速度。

Polynomial Kernel

我们刚刚通过一个特殊的二次多项式导出了相对应的kernel，其实二次多项式的kernel形式是多种的，

例如，相应系数的放缩构成完全平方公式等。下面列举了几种常用的二次多项式kernel形式：

$$\Phi_2(\mathbf{x}) = (1, x_1, \ldots, x_d, x_1^2, \ldots, x_d^2) \Leftrightarrow K_{\Phi_2}(\mathbf{x}, \mathbf{x}') = 1 + \mathbf{x}^T\mathbf{x}' + (\mathbf{x}^T\mathbf{x}')^2$$

$$\Phi_2(\mathbf{x}) = (1, \sqrt{2}x_1, \ldots, \sqrt{2}x_d, x_1^2, \ldots, x_d^2) \Leftrightarrow K_2(\mathbf{x}, \mathbf{x}') = 1 + 2\mathbf{x}^T\mathbf{x}' + (\mathbf{x}^T\mathbf{x}')^2$$

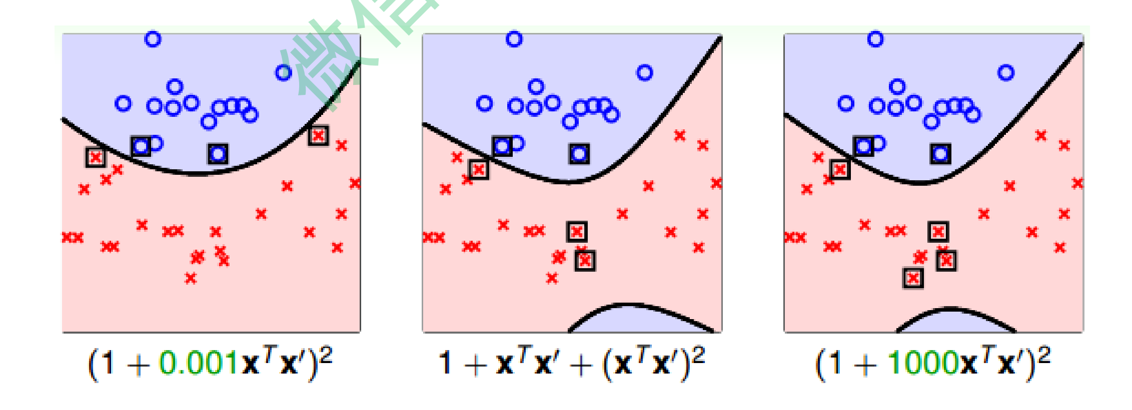
$$\Phi_2(\mathbf{x}) = (1, \sqrt{2\gamma}x_1, \ldots, \sqrt{2\gamma}x_d, \gamma x_1^2, \ldots, \gamma x_d^2) \Leftrightarrow K_2(\mathbf{x}, \mathbf{x}') = 1 + 2\gamma\mathbf{x}^T\mathbf{x}' + \gamma^2(\mathbf{x}^T\mathbf{x}')^2$$

比较一下，第一种$\Phi_2(x)$（蓝色标记）和第三种$\Phi_2(x)$（绿色标记）从某种角度来说是一样的，因为都是二次转换，对应到同一个z空间。但是，它们系数不同，内积就会有差异，那么就代表有不同的距离，最终可能会得到不同的SVM margin。所以，系数不同，可能会得到不同的SVM分界线。通常情况下，第三种$\Phi_2(x)$（绿色标记）简单一些，更加常用。

$$K_2(\mathbf{x}, \mathbf{x}') = (1 + \gamma\mathbf{x}^T\mathbf{x}')^2 \text{ with } \gamma > 0$$

- K_2: somewhat '**easier**' to calculate than K_{Φ_2}
- Φ_2 and Φ_2: equivalent **power**,
  different inner product $\Longrightarrow$ different **geometry**

不同的转换，对应到不同的几何距离，得到不同的距离，这是什么意思呢？举个例子，对于我们之前介绍的一般的二次多项式kernel，它的SVM margin和对应的SV如下图（中）所示。对于上面介绍的完全平方公式形式，自由度$\gamma = 0.001$，它的SVM margin和对应的SV如下图（左）所示。比较发现，这种SVM margin比较简单一些。对于自由度$\gamma = 1000$，它的SVM margin和对应的SV如下图（右）所示。与前两种比较，margin和SV都有所不同。



通过改变不同的系数，得到不同的SVM margin和SV，如何选择正确的kernel，非常重要。

归纳一下，引入$\zeta \geq 0$和$\gamma > 0$，对于Q次多项式一般的kernel形式可表示为：



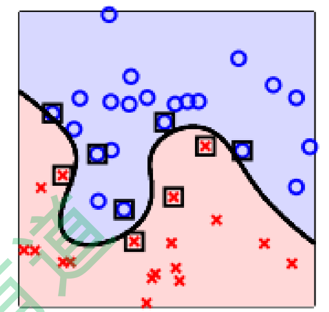
$$
\begin{aligned}
K_2(\mathbf{x},\mathbf{x}') &= (\zeta+\gamma\mathbf{x}^T\mathbf{x}')^2 \text{ with } \gamma>0, \zeta\geq 0 \\
K_3(\mathbf{x},\mathbf{x}') &= (\zeta+\gamma\mathbf{x}^T\mathbf{x}')^3 \text{ with } \gamma>0, \zeta\geq 0 \\
&\vdots \\
K_Q(\mathbf{x},\mathbf{x}') &= (\zeta+\gamma\mathbf{x}^T\mathbf{x}')^Q \text{ with } \gamma>0, \zeta\geq 0
\end{aligned}
$$

所以，使用高阶的多项式kernel有两个优点：

- **得到最大SVM margin，SV数量不会太多，分类面不会太复杂，防止过拟合，减少复杂度**
- **计算过程避免了对$\hat{d}$的依赖，大大简化了计算量。**

- embeds $\mathbf{\Phi}_Q$ specially with parameters (γ,ζ)
- allows computing large-margin polynomial classification without dependence on $\tilde{d}$

SVM + Polynomial Kernel: Polynomial SVM



10-th order polynomial
with margin 0.1

顺便提一下，当多项式阶数Q=1时，那么对应的kernel就是线性的，即本系列课程第一节课所介绍的内容。对于linear kernel，计算方法是简单的，而且也是我们解决SVM问题的首选。还记得机器学习基石课程中介绍的奥卡姆剃刀定律（Occam's Razor）吗？

Gaussian Kernel

刚刚我们介绍的Q阶多项式kernel的阶数是有限的，即特征转换的$\hat{d}$是有限的。但是，如果是无限多维的转换$\Phi(x)$，是否还能通过kernel的思想，来简化SVM的计算呢？答案是肯定的。

先举个例子，简单起见，假设原空间是一维的，只有一个特征x，我们构造一个kernel function为高斯函数：

$$K(x,x')=e^{-(x-x')^2}$$

构造的过程正好与二次多项式kernel的相反，利用反推法，先将上式分解并做泰勒展开：



$$
\begin{aligned}
\text{when } \mathbf{x} = (x),\ K(x, x') &= \exp(-(x - x')^2) \\
&= \exp(-(x)^2)\exp(-(x')^2)\exp(2xx') \\
&\overset{\text{Taylor}}{=} \exp(-(x)^2)\exp(-(x')^2)\left(\sum_{i=0}^{\infty}\frac{(2xx')^i}{i!}\right) \\
&= \sum_{i=0}^{\infty}\left(\exp(-(x)^2)\exp(-(x')^2)\sqrt{\frac{2^i}{i!}}\sqrt{\frac{2^i}{i!}}(x)^i(x')^i\right) \\
&= \Phi(x)^T\Phi(x')
\end{aligned}
$$

with infinite dimensional $\Phi(x) = \exp(-x^2) \cdot \left(1, \sqrt{\frac{2}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \ldots\right)$

将构造的K(x,x')推导展开为两个$\Phi(x)$和$\Phi(x')$的乘积，其中:

$$\Phi(x) = e^{-x^2} \cdot (1, \sqrt{\frac{2}{1!}}x, \sqrt{\frac{2^2}{2!}}x^2, \cdots)$$

通过反推，我们得到了$\Phi(x)$，$\Phi(x)$是无限多维的，它就可以当成特征转换的函数，且$\hat{d}$是无限的。这种$\Phi(x)$得到的核函数即为Gaussian kernel。

更一般地，对于原空间不止一维的情况（d>1），引入缩放因子$\gamma > 0$，它对应的Gaussian kernel表达式为:

$$K(x, x') = e^{-\gamma||x-x'||^2}$$

那么引入了高斯核函数，将有限维度的特征转换拓展到无限的特征转换中。根据本节课上一小节的内容，由K，计算得到α_n和b，进而得到矩g_{SVM}。将其中的核函数K用高斯核函数代替，得到:

$$g_{SVM}(x) = sign(\sum_{SV}\alpha_n y_n K(x_n, x) + b) = sign(\sum_{SV}\alpha_n y_n e^{(-\gamma||x-x_n||^2)} + b)$$

通过上式可以看出，g_{SVM}有n个高斯函数线性组合而成，其中n是SV的个数。而且，每个高斯函数的中心都是对应的SV。通常我们也把高斯核函数称为径向基函数（Radial Basis Function, RBF）。

$$
\begin{aligned}
g_{\text{SVM}}(\mathbf{x}) &= \text{sign}\left(\sum_{\text{SV}}\alpha_n y_n K(\mathbf{x}_n, \mathbf{x}) + b\right) \\
&= \text{sign}\left(\sum_{\text{SV}}\alpha_n y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right) + b\right)
\end{aligned}
$$

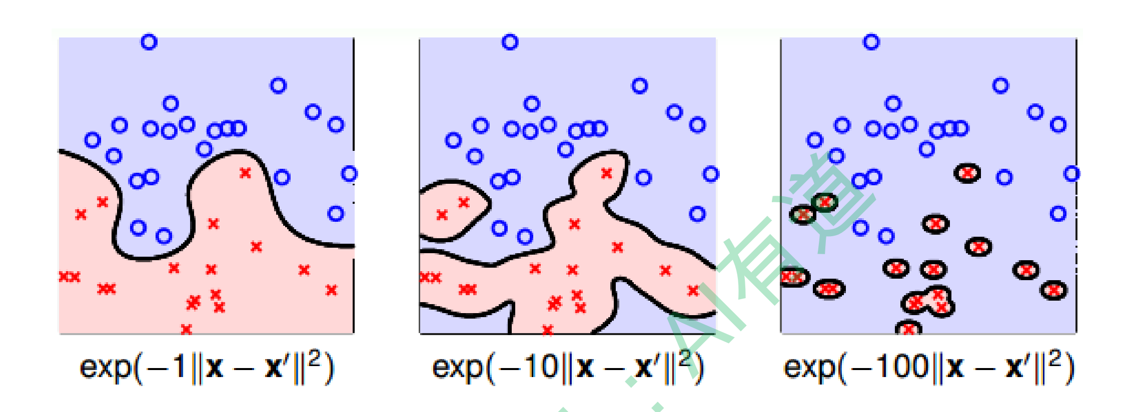
- linear combination of Gaussians centered at SVs $\mathbf{x}_n$
- also called Radial Basis Function (RBF) kernel

总结一下，kernel SVM可以获得large-margin的hyperplanes，并且可以通过高阶的特征转换使E_{in}尽可能地小。kernel的引入大大简化了dual SVM的计算量。而且，Gaussian kernel能将特征转换扩展到无限维，并使用有限个SV数量的高斯函数构造出矩g_{SVM}。

	large-margin hyperplanes + higher-order transforms with kernel trick
#	not many
boundary	sophisticated

- transformed vector $\mathbf{z} = \mathbf{\Phi}(\mathbf{x}) \Longrightarrow$ efficient kernel $K(\mathbf{x}, \mathbf{x}')$
- store optimal $\mathbf{w} \Longrightarrow$ store a few SVs and α_n

值得注意的是，缩放因子γ取值不同，会得到不同的高斯核函数，hyperplanes不同，分类效果也有很大的差异。举个例子，γ分别取1, 10, 100时对应的分类效果如下：

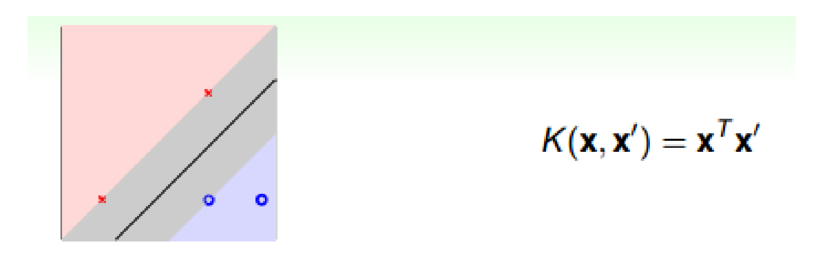


从图中可以看出，当γ比较小的时候，分类线比较光滑，当γ越来越大的时候，分类线变得越来越复杂和扭曲，直到最后，分类线变成一个个独立的小区域，像小岛一样将每个样本单独包起来了。为什么会出现这种区别呢？这是因为γ越大，其对应的高斯核函数越尖瘦，那么有限个高斯核函数的线性组合就比较离散，分类效果并不好。所以，SVM也会出现过拟合现象，γ的正确选择尤为重要，不能太大。

Comparison of Kernels

目前为止，我们已经介绍了几种kernel，下面来对几种kernel进行比较。

首先，Linear Kernel是最简单最基本的核，平面上对应一条直线，三维空间里对应一个平面。Linear Kernel可以使用上一节课介绍的Dual SVM中的QP直接计算得到。



Linear Kernel的优点是计算简单、快速，可以直接使用QP快速得到参数值，而且从视觉上分类效果非常直观，便于理解；缺点是如果数据不是线性可分的情况，Linear Kernel就不能使用了。

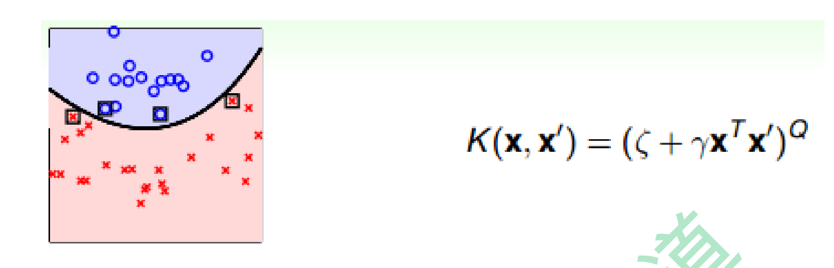
Cons

- restricted
  —**not always separable?!**

Pros

- safe—**linear first, remember? :-)**
- fast—with **special QP solver** in primal
- very explainable—**w and SVs** say something

然后，Polynomial Kernel的hyperplanes是由多项式曲线构成。



Polynomial Kernel的优点是阶数Q可以灵活设置，相比linear kernel限制更少，更贴近实际样本分布；缺点是当Q很大时，K的数值范围波动很大，而且参数个数较多，难以选择合适的值。



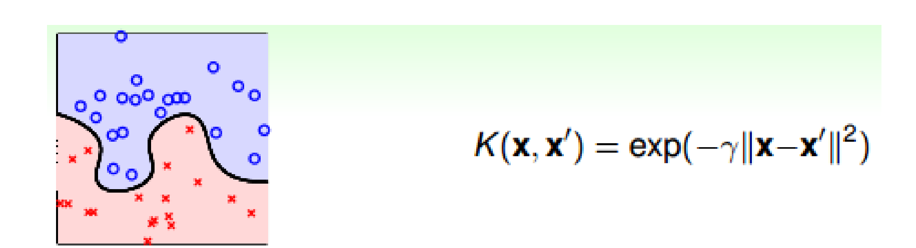
Cons

- **numerical difficulty** for large Q
  - $|\zeta + \gamma \mathbf{x}^T \mathbf{x}'| < 1$: $K \to 0$
  - $|\zeta + \gamma \mathbf{x}^T \mathbf{x}'| > 1$: $K \to$ big
- three parameters (γ, ζ, Q)
  —**more difficult to select**

Pros

- **less restricted** than linear
- strong physical control
  —'knows' **degree Q**

对于Gaussian Kernel，表示为高斯函数形式。



Gaussian Kernel的优点是边界更加复杂多样，能最准确地区分数据样本，数值计算K值波动较小，而且只有一个参数，容易选择；缺点是由于特征转换到无限维度中，w没有求解出来，计算速度要低于linear kernel，而且可能会发生过拟合。

Cons

- mysterious—no w
- slower than linear
- too powerful?!

Pros

- more powerful than linear/poly.
- bounded—less numerical difficulty than poly.
- one parameter only—easier to select than poly.

除了这三种kernel之外，我们还可以使用其它形式的kernel。首先，我们考虑kernel是什么？实际上kernel代表的是两笔资料x和x'，特征变换后的相似性即内积。但是不能说任何计算相似性的函数都可以是kernel。有效的kernel还需满足几个条件：

- **K是对称的**
- **K是半正定的**

这两个条件不仅是必要条件，同时也是充分条件。所以，只要我们构造的K同时满足这两个条件，那它就是一个有效的kernel。这被称为Mercer 定理。事实上，构造一个有效的kernel是比较困难的。

- kernel represents special similarity: $\Phi(\mathbf{x})^T\Phi(\mathbf{x}')$
- any similarity $\Longrightarrow$ valid kernel? not really
- necessary & sufficient conditions for valid kernel: Mercer's condition
  - symmetric
  - let $k_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, the matrix K

$$
= \begin{bmatrix} \Phi(\mathbf{x}_1)^T\Phi(\mathbf{x}_1) & \Phi(\mathbf{x}_1)^T\Phi(\mathbf{x}_2) & \dots & \Phi(\mathbf{x}_1)^T\Phi(\mathbf{x}_N) \\ \Phi(\mathbf{x}_2)^T\Phi(\mathbf{x}_1) & \Phi(\mathbf{x}_2)^T\Phi(\mathbf{x}_2) & \dots & \Phi(\mathbf{x}_2)^T\Phi(\mathbf{x}_N) \\ \dots & \dots & \dots & \dots \\ \Phi(\mathbf{x}_N)^T\Phi(\mathbf{x}_1) & \Phi(\mathbf{x}_N)^T\Phi(\mathbf{x}_2) & \dots & \Phi(\mathbf{x}_N)^T\Phi(\mathbf{x}_N) \end{bmatrix}
$$

$$
= [\,\mathbf{z}_1 \;\; \mathbf{z}_2 \;\; \dots \;\; \mathbf{z}_N\,]^T [\,\mathbf{z}_1 \;\; \mathbf{z}_2 \;\; \dots \;\; \mathbf{z}_N\,]
$$

$$
= \mathrm{ZZ}^T \text{ must always be positive semi-definite}
$$

总结

本节课主要介绍了Kernel Support Vector Machine。首先，我们将特征转换和计算内积的操作合并到一起，消除了$\hat{d}$的影响，提高了计算速度。然后，分别推导了Polynomial Kernel和Gaussian Kernel，并列举了各自的优缺点并做了比较。对于不同的问题，应该选择合适的核函数进行求解，以达到最佳的分类效果。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程



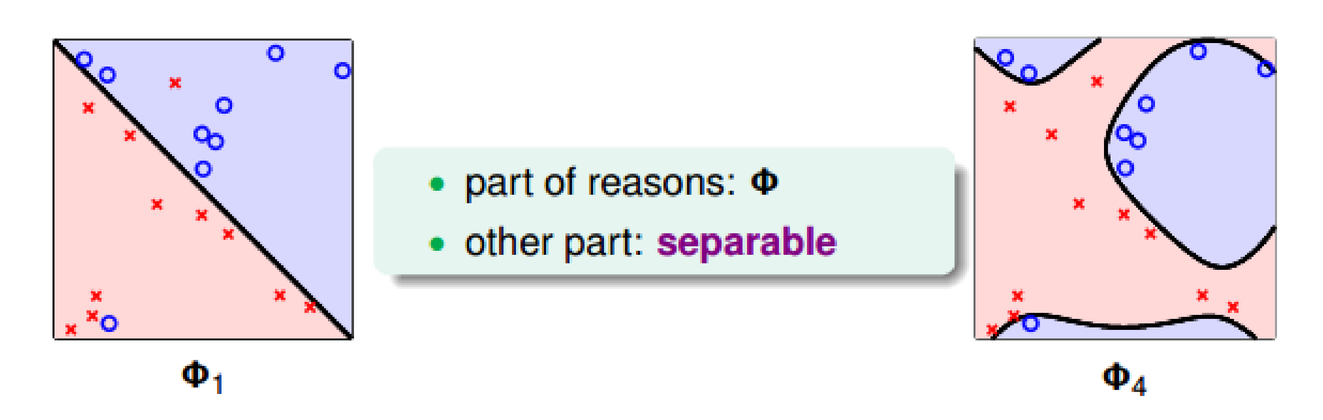
林轩田《机器学习技法》课程笔记4 -- Soft-Margin Support Vector Machine

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了Kernel SVM。先将特征转换和计算内积这两个步骤合并起来，简化计算、提高计算速度，再用Dual SVM的求解方法来解决。Kernel SVM不仅能解决简单的线性分类问题，也可以求解非常复杂甚至是无限多维的分类问题，关键在于核函数的选择，例如线性核函数、多项式核函数和高斯核函数等等。但是，我们之前讲的这些方法都是Hard-Margin SVM，即必须将所有的样本都分类正确才行。这往往需要更多更复杂的特征转换，甚至造成过拟合。本节课将介绍一种Soft-Margin SVM，目的是让分类错误的点越少越好，而不是必须将所有点分类正确，也就是允许有noise存在。这种做法很大程度上不会使模型过于复杂，不会造成过拟合，而且分类效果是令人满意的。

Motivation and Primal Problem

上节课我们说明了一点，就是SVM同样可能会造成overfit。原因有两个，一个是由于我们的SVM模型（即kernel）过于复杂，转换的维度太多，过于powerful了；另外一个是由于我们坚持要将所有的样本都分类正确，即不允许错误存在，造成模型过于复杂。如下图所示，左边的图Φ_1是线性的，虽然有几个点分类错误，但是大部分都能完全分开。右边的图Φ_4是四次多项式，所有点都分类正确了，但是模型比较复杂，可能造成过拟合。直观上来说，左边的图是更合理的模型。



如何避免过拟合？方法是允许有分类错误的点，即把某些点当作是noise，放弃这些



noise点，但是尽量让这些noise个数越少越好。回顾一下我们在机器学习基石笔记中介绍的pocket算法，pocket的思想不是将所有点完全分开，而是找到一条分类线能让分类错误的点最少。而Hard-Margin SVM的目标是将所有点都完全分开，不允许有错误点存在。为了防止过拟合，我们可以借鉴pocket的思想，即允许有犯错误的点，目标是让这些点越少越好。

pocket

$$\min_{b,\mathbf{w}} \sum_{n=1}^{N} [\![y_n \neq \text{sign}(\mathbf{w}^T\mathbf{z}_n + b)]\!]$$

hard-margin SVM

$$\min_{b,\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w}$$

$$\text{s.t.} \quad y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1 \text{ for all } n$$

为了引入允许犯错误的点，我们将Hard-Margin SVM的目标和条件做一些结合和修正，转换为如下形式：

$$\text{combination:} \quad \min_{b,\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \cdot \sum_{n=1}^{N} [\![y_n \neq \text{sign}(\mathbf{w}^T\mathbf{z}_n + b)]\!]$$

$$\text{s.t.} \quad y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1 \text{ for } \textbf{correct } n$$

$$y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq -\infty \text{ for } \textbf{incorrect } n$$

修正后的条件中，对于分类正确的点，仍需满足$y_n(w^Tz_n+b)\geq 1$，而对于noise点，满足$y_n(w^Tz_n+b)\geq -\infty$，即没有限制。修正后的目标除了$\frac{1}{2}w^Tw$项，还添加了$y_n \neq sign(w^Tz_n+b)$，即noise点的个数。参数C的引入是为了权衡目标第一项和第二项的关系，即权衡large margin和noise tolerance的关系。

我们再对上述的条件做修正，将两个条件合并，得到：

$$\min_{b,\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \cdot \sum_{n=1}^{N} [\![y_n \neq \text{sign}(\mathbf{w}^T\mathbf{z}_n + b)]\!]$$

$$\text{s.t.} \quad y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1 - \infty \cdot [\![y_n \neq \text{sign}(\mathbf{w}^T\mathbf{z}_n + b)]\!]$$

这个式子存在两个不足的地方。首先，最小化目标中第二项是非线性的，不满足QP的条件，所以无法使用dual或者kernel SVM来计算。然后，对于犯错误的点，有的离边界很近，即error小，而有的离边界很远，error很大，上式的条件和目标没有区分sma

error和large error。这种分类效果是不完美的。

- $[\![\cdot]\!]$: non-linear, **not QP anymore :-(**
  —what about dual? kernel?
- cannot distinguish **small error** (slightly away from fat boundary)
  or **large error** (a...w...a...y... from fat boundary)

为了改正这些不足，我们继续做如下修正：

- record '**margin violation**' by ξ_n—**linear constraints**
- penalize with **margin violation** instead of **error count**
  —**quadratic objective**

$$\text{soft-margin SVM: } \min_{b,\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \cdot \sum_{n=1}^{N} \xi_n$$

$$\text{s.t.} \quad y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1 - \xi_n \text{ and } \xi_n \geq 0 \text{ for all } n$$

修正后的表达式中，我们引入了新的参数ξ_n来表示每个点犯错误的程度值，$\xi_n \geq 0$。通过使用error值的大小代替是否有error，让问题变得易于求解，满足QP形式要求。这种方法类似于我们在机器学习基石笔记中介绍的0/1 error和squared error。这种soft-margin SVM引入新的参数ξ。

至此，最终的Soft-Margin SVM的目标为：

$$min(b,w,\xi)\ \frac{1}{2}w^Tw + C \cdot \sum_{n=1}^{N} \xi_n$$

条件是：

$$y_n(w^Tz_n + b) \geq 1 - \xi_n$$

$$\xi_n \geq 0$$

其中，ξ_n表示每个点犯错误的程度，$\xi_n = 0$，表示没有错误，ξ_n越大，表示错误越大，即点距离边界（负的）越大。参数C表示尽可能选择宽边界和尽可能不要犯错两者之间的权衡，因为边界宽了，往往犯错误的点会增加。large C表示希望得到更少的分类错误，即不惜选择窄边界也要尽可能把更多点正确分类；small C表示希望得到更宽的边界，即不惜增加错误点个数也要选择更宽的分类边界。

与之对应的QP问题中，由于新的参数ξ_n的引入，总共参数个数为$\hat{d} + 1 + N$，限制

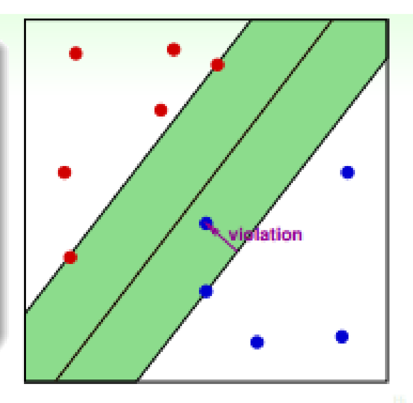


条件添加了$\xi_n \geq 0$，则总条件个数为2N。

- record 'margin violation' by ξ_n
- penalize with margin violation

$$\min_{b,\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \cdot \sum_{n=1}^{N} \xi_n$$

$$\text{s.t.} \quad y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1 - \xi_n \text{ and } \xi_n \geq 0 \text{ for all } n$$



- parameter C: trade-off of large margin & margin violation
  - large C: want less margin violation
  - small C: want large margin
- QP of $\tilde{d} + 1 + N$ variables, $2N$ constraints

Dual Problem

接下来，我们将推导Soft-Margin SVM的对偶dual形式，从而让QP计算更加简单，并便于引入kernel算法。首先，我们把Soft-Margin SVM的原始形式写出来：

$$\text{primal:} \quad \min_{b,\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \cdot \sum_{n=1}^{N} \xi_n$$

$$\text{s.t.} \quad y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1 - \xi_n \text{ and } \xi_n \geq 0 \text{ for all } n$$

然后，跟我们在第二节课中介绍的Hard-Margin SVM做法一样，构造一个拉格朗日函数。因为引入了ξ_n，原始问题有两类条件，所以包含了两个拉格朗日因子α_n和β_n。拉格朗日函数可表示为如下形式：

Lagrange function with Lagrange multipliers α_n and β_n

$$\mathcal{L}(b, \mathbf{w}, \xi, \alpha, \beta) = \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \cdot \sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \cdot (1 - \xi_n - y_n(\mathbf{w}^T\mathbf{z}_n + b)) + \sum_{n=1}^{N} \beta_n \cdot (-\xi_n)$$



接下来，我们跟第二节课中的做法一样，利用Lagrange dual problem，将Soft-Margin SVM问题转换为如下形式：

$$\max_{\alpha_n \ge 0,\ \beta_n \ge 0} \left(\min_{b,\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \cdot \sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \cdot (1 - \xi_n - y_n(\mathbf{w}^T\mathbf{z}_n + b)) + \sum_{n=1}^{N} \beta_n \cdot (-\xi_n) \right)$$

根据之前介绍的KKT条件，我们对上式进行简化。上式括号里面的是对拉格朗日函数$L(b, w, \xi, \alpha, \beta)$计算最小值。那么根据梯度下降算法思想：最小值位置满足梯度为零。

我们先对ξ_n做偏微分：

$$\frac{\partial L}{\partial \xi_n} = 0 = C - \alpha_n - \beta_n$$

根据上式，得到$\beta_n = C - \alpha_n$，因为有$\beta_n \ge 0$，所以限制$0 \le \alpha_n \le C$。将$\beta_n = C - \alpha_n$代入到dual形式中并化简，我们发现β_n和ξ_n都被消去了：

$$\max_{0 \le \alpha_n \le C,\ \beta_n = C - \alpha_n} \left(\min_{b,\mathbf{w}} \frac{1}{2}\mathbf{w}^T\mathbf{w} + \sum_{n=1}^{N} \alpha_n (1 - y_n(\mathbf{w}^T\mathbf{z}_n + b)) \right)$$

这个形式跟Hard-Margin SVM中的dual形式是基本一致的，只是条件不同。那么，我们分别令拉个朗日函数L对b和w的偏导数为零，分别得到：

$$\sum_{n=1}^{N} \alpha_n y_n = 0$$

$$w = \sum_{n=1}^{N} \alpha_n y_n z_n$$

经过化简和推导，最终标准的Soft-Margin SVM的Dual形式如下图所示：



$$\min_{\alpha} \quad \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}\alpha_n\alpha_m y_n y_m \mathbf{z}_n^T\mathbf{z}_m - \sum_{n=1}^{N}\alpha_n$$

$$\text{subject to} \quad \sum_{n=1}^{N} y_n\alpha_n = 0;$$

$$0 \le \alpha_n \le C, \text{for } n = 1, 2, \ldots, N;$$

$$\text{implicitly} \quad \mathbf{w} = \sum_{n=1}^{N}\alpha_n y_n \mathbf{z}_n;$$

$$\beta_n = C - \alpha_n, \text{for } n = 1, 2, \ldots, N$$

—only difference to hard-margin: upper bound on α_n

Soft-Margin SVM Dual与Hard-Margin SVM Dual基本一致，只有一些条件不同。Hard-Margin SVM Dual中$\alpha_n \geq 0$，而Soft-Margin SVM Dual中$0 \leq \alpha_n \leq C$，且新的拉格朗日因子$\beta_n = C - \alpha_n$。在QP问题中，Soft-Margin SVM Dual的参数α_n同样是N个，但是，条件由Hard-Margin SVM Dual中的N+1个变成2N+1个，这是因为多了N个α_n的上界条件。

对于Soft-Margin SVM Dual这部分推导不太清楚的同学，可以看下第二节课的笔记：台湾大学林轩田机器学习技法课程学习笔记2 -- Dual Support Vector Machine

Messages behind Soft-Margin SVM

推导完Soft-Margin SVM Dual的简化形式后，就可以利用QP，找到Q，p，A，c对应的值，用软件工具包得到α_n的值。或者利用核函数的方式，同样可以简化计算，优化分类效果。Soft-Margin SVM Dual计算α_n的方法过程与Hard-Margin SVM Dual的过程是相同的。

Kernel Soft-Margin SVM Algorithm

1. $q_{n,m} = y_n y_m K(\mathbf{x}_n, \mathbf{x}_m)$; $\mathbf{p} = -\mathbf{1}_N$; $(\mathrm{A}, \mathbf{c})$ for equ./lower-bound/**upper-bound** constraints
2. $\boldsymbol{\alpha} \leftarrow \mathrm{QP}(\mathrm{Q_D}, \mathbf{p}, \mathrm{A}, \mathbf{c})$
3. $b \leftarrow ?$
4. return SVs and their α_n as well as b such that for new $\mathbf{x}$,

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV indices } n}\alpha_n y_n K(\mathbf{x}_n, \mathbf{x}) + b\right)$$



但是如何根据α_n的值计算b呢？在Hard-Margin SVM Dual中，有complementary slackness条件：$\alpha_n(1-y_n(w^Tz_n+b))=0$，找到SV，即$\alpha_s>0$的点，计算得到$b=y_s-w^Tz_s$。

那么，在Soft-Margin SVM Dual中，相应的complementary slackness条件有两个（因为两个拉格朗日因子α_n和β_n）：

$$\alpha_n(1-\xi_n-y_n(w^Tz_n+b))=0$$

$$\beta_n\xi_n=(C-\alpha_n)\xi=0$$

找到SV，即$\alpha_s>0$的点，由于参数ξ_n的存在，还不能完全计算出b的值。根据第二个complementary slackness条件，如果令$C-\alpha_n\neq 0$，即$\alpha_n\neq C$，则一定有$\xi_n=0$，代入到第一个complementary slackness条件，即可计算得到$b=y_s-w^Tz_s$。我们把$0<\alpha_s<C$的点称为free SV。引入核函数后，b的表达式为：

$$b=y_s-\sum_{SV}\alpha_ny_nK(x_n,x_s)$$

上面求解b提到的一个假设是$\alpha_s<C$，这个假设是否一定满足呢？如果没有free SV，所有α_s大于零的点都满足$\alpha_s=C$怎么办？一般情况下，至少存在一组SV使$\alpha_s<C$的概率是很大的。如果出现没有free SV的情况，那么b通常会由许多不等式条件限制取值范围，值是不确定的，只要能找到其中满足KKT条件的任意一个b值就可以了。这部分细节比较复杂，不再赘述。

hard-margin SVM

complementary slackness:

$$\alpha_n(1-y_n(\mathbf{w}^T\mathbf{z}_n+b))=0$$

- SV ($\alpha_s>0$)
  $\Rightarrow b=y_s-\mathbf{w}^T\mathbf{z}_s$

soft-margin SVM

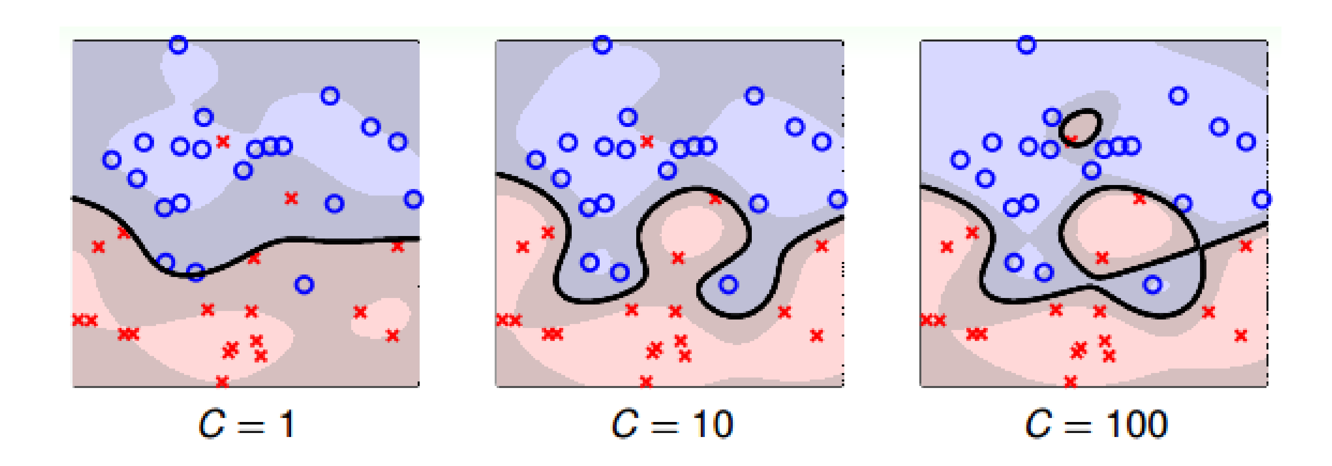
complementary slackness:

$$\alpha_n(1-\xi_n-y_n(\mathbf{w}^T\mathbf{z}_n+b))=0$$
$$(C-\alpha_n)\xi_n=0$$

- SV ($\alpha_s>0$)
  $\Rightarrow b=y_s-y_s\xi_s-\mathbf{w}^T\mathbf{z}_s$
- free ($\alpha_s<C$)
  $\Rightarrow \xi_s=0$

接下来，我们看看C取不同的值对margin的影响。例如，对于Soft-Margin Gaussian SVM，C分别取1，10，100时，相应的margin如下图所示：





从上图可以看出，C=1时，margin比较粗，但是分类错误的点也比较多，当C越来越大的时候，margin越来越细，分类错误的点也在减少。正如前面介绍的，C值反映了margin和分类正确的一个权衡。C越小，越倾向于得到粗的margin，宁可增加分类错误的点；C越大，越倾向于得到高的分类正确率，宁可margin很细。我们发现，当C值很大的时候，虽然分类正确率提高，但很可能把noise也进行了处理，从而可能造成过拟合。也就是说Soft-Margin Gaussian SVM同样可能会出现过拟合现象，所以参数(γ, C)的选择非常重要。

我们再来看看α_n取不同值是对应的物理意义。已知$0 \leq \alpha_n \leq C$满足两个complementary slackness条件：

$$\alpha_n(1-\xi_n-y_n(w^Tz_n+b))=0$$

$$\beta_n\xi_n=(C-\alpha_n)\xi=0$$

若$\alpha_n=0$，得$\xi_n=0$。$\xi_n=0$表示该点没有犯错，$\alpha_n=0$表示该点不是SV。所以对应的点在margin之外（或者在margin上），且均分类正确。

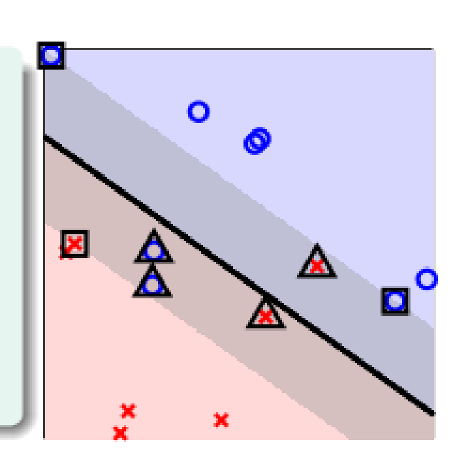
若$0<\alpha_n<C$，得$\xi_n=0$，且$y_n(w^Tz_n+b)=1$。$\xi_n=0$表示该点没有犯错，$y_n(w^Tz_n+b)=1$表示该点在margin上。这些点即free SV，确定了b的值。

若$\alpha_n=C$，不能确定ξ_n是否为零，且得到$1-y_n(w^Tz_n+b)=\xi_n$，这个式表示该点偏离margin的程度，ξ_n越大，偏离margin的程度越大。只有当$\xi_n=0$时，该点落在margin上。所以这种情况对应的点在margin之内负方向（或者在margin上），有分类正确也有分类错误的。这些点称为bounded SV。

所以，在Soft-Margin SVM Dual中，根据α_n的取值，就可以推断数据点在空间的分布情况。

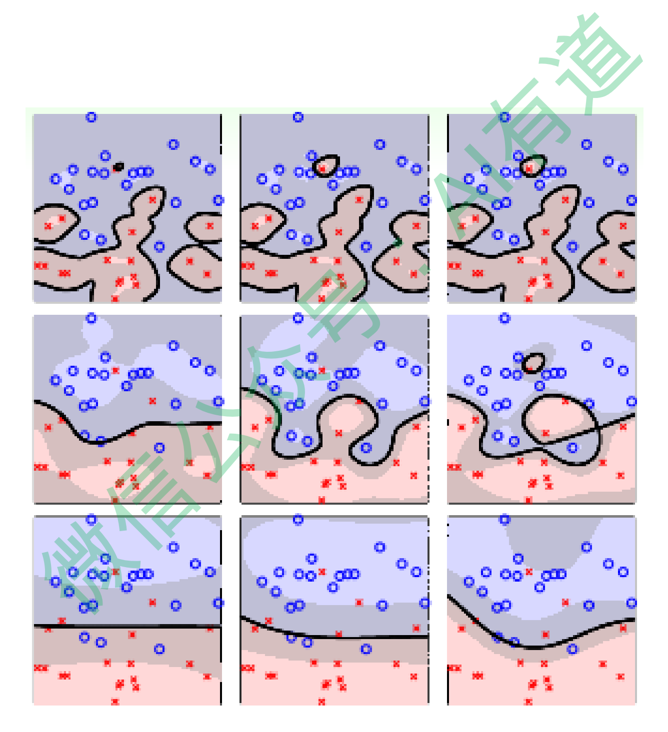


- non SV ($0 = \alpha_n$): $\xi_n = 0$, 'away from'/on fat boundary
- □ free SV ($0 < \alpha_n < C$): $\xi_n = 0$, on fat boundary, locates b
- △ bounded SV ($\alpha_n = C$): ξ_n = violation amount, 'violate'/on fat boundary



Model Selection

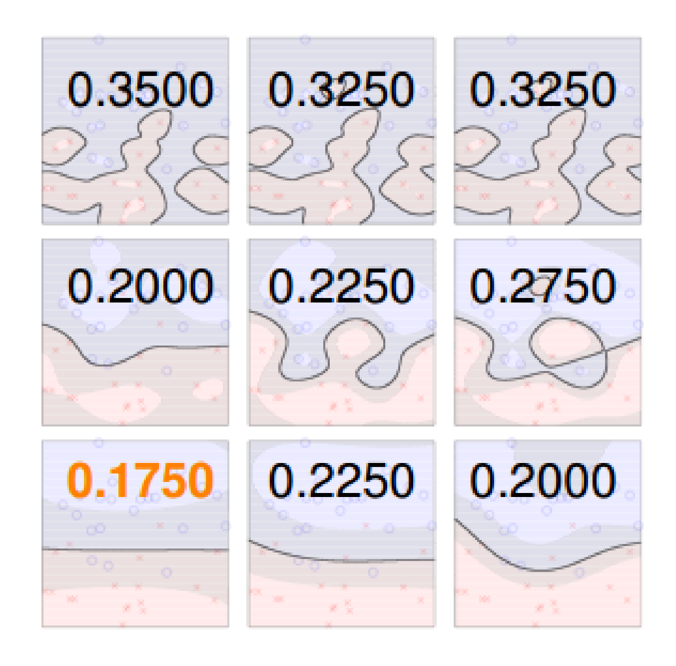
在Soft-Margin SVM Dual中，kernel的选择、C等参数的选择都非常重要，直接影响分类效果。例如，对于Gaussian SVM，不同的参数(C,γ)，会得到不同的margin，如下图所示。



其中横坐标是C逐渐增大的情况，纵坐标是γ逐渐增大的情况。不同的(C,γ)组合，margin的差别很大。那么如何选择最好的(C,γ)等参数呢？最简单最好用的工具就是validation。

validation我们在机器学习基石课程中已经介绍过，只需要将由不同(C,γ)等参数得到的模型在验证集上进行cross validation，选取E_{cv}最小的对应的模型就可以了。例如上图中各种(C,γ)组合得到的E_{cv}如下图所示：





因为左下角的$E_{cv}(C,\gamma)$最小，所以就选择该(C,γ)对应的模型。通常来说，$E_{cv}(C,\gamma)$并不是(C,γ)的连续函数，很难使用最优化选择（例如梯度下降）。一般做法是选取不同的离散的(C,γ)值进行组合，得到最小的$E_{cv}(C,\gamma)$，其对应的模型即为最佳模型。这种算法就是我们之前在机器学习基石中介绍过的V-Fold cross validation，在SVM中使用非常广泛。

V-Fold cross validation的一种极限就是Leave-One-Out CV，也就是验证集只有一个样本。对于SVM问题，它的验证集Error满足：

$$E_{loocv} \leq \frac{SV}{N}$$

也就是说留一法验证集Error大小不超过支持向量SV占所有样本的比例。下面做简单的证明。令样本总数为N，对这N个点进行SVM分类后得到margin，假设第N个点(x_N,y_N)的$\alpha_N=0$，不是SV，即远离margin（正距离）。这时候，如果我们只使用剩下的N-1个点来进行SVM分类，那么第N个点(x_N,y_N)必然是分类正确的点，所得的SVM margin跟使用N个点的到的是完全一致的。这是因为我们假设第N个点是non-SV，对SV没有贡献，不影响margin的位置和形状。所以前N-1个点和N个点得到的margin是一样的。

那么，对于non-SV的点，它的$g^-=g$，即对第N个点，它的Error必然为零：

$$e_{non-SV}=err(g^-,non-SV)=err(g,non-SV)=0$$

另一方面，假设第N个点$\alpha_N\neq 0$，即对于SV的点，它的Error可能是0，也可能是1，必然有：

$$e_{SV}\leq 1$$



综上所述，即证明了$E_{loocv} \leq \frac{SV}{N}$。这符合我们之前得到的结论，即只有SV影响margin，non-SV对margin没有任何影响，可以舍弃。

SV的数量在SVM模型选择中也是很重要的。一般来说，SV越多，表示模型可能越复杂，越有可能会造成过拟合。所以，通常选择SV数量较少的模型，然后在剩下的模型中使用cross-validation，比较选择最佳模型。

总结

本节课主要介绍了Soft-Margin SVM。我们的出发点是与Hard-Margin SVM不同，不一定要将所有的样本点都完全分开，允许有分类错误的点，而使margin比较宽。然后，我们增加了ξ_n作为分类错误的惩罚项，根据之前介绍的Dual SVM，推导出了Soft-Margin SVM的QP形式。得到的α_n除了要满足大于零，还有一个上界C。接着介绍了通过α_n值的大小，可以将数据点分为三种：non-SVs，free SVs，bounded SVs，这种更清晰的物理解释便于数据分析。最后介绍了如何选择合适的SVM模型，通常的办法是cross-validation和利用SV的数量进行筛选。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程



林轩田《机器学习技法》课程笔记5 -- Kernel Logistic Regression

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了Soft-Margin SVM，即如果允许有分类错误的点存在，那么在原来的Hard-Margin SVM中添加新的惩罚因子C，修正原来的公式，得到新的α_n值。最终的到的α_n有个上界，上界就是C。Soft-Margin SVM权衡了large-margin和error point之前的关系，目的是在尽可能犯更少错误的前提下，得到最大分类边界。本节课将把Soft-Margin SVM和我们之前介绍的Logistic Regression联系起来，研究如何使用kernel技巧来解决更多的问题。

Soft-Margin SVM as Regularized Model

先复习一下我们已经介绍过的内容，我们最早开始讲了Hard-Margin Primal的数学表达式，然后推导了Hard-Margin Dual形式。后来，为了允许有错误点的存在（或者noise），也为了避免模型过于复杂化，造成过拟合，我们建立了Soft-Margin Primal的数学表达式，并引入了新的参数C作为权衡因子，然后也推导了其Soft-Margin Dual形式。因为Soft-Margin Dual SVM更加灵活、便于调整参数，所以在实际应用中，使用Soft-Margin Dual SVM来解决分类问题的情况更多一些。

Hard-Margin Primal

$$\min_{b,\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w}$$
$$\text{s.t.} \quad y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1$$

Soft-Margin Primal

$$\min_{b,\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n$$
$$\text{s.t.} \quad y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1 - \xi_n, \xi_n \geq 0$$

Hard-Margin Dual

$$\min_{\alpha} \quad \frac{1}{2}\alpha^T\mathrm{Q}\alpha - \mathbf{1}^T\alpha$$
$$\text{s.t.} \quad \mathbf{y}^T\alpha = 0$$
$$0 \leq \alpha_n$$

Soft-Margin Dual

$$\min_{\alpha} \quad \frac{1}{2}\alpha^T\mathrm{Q}\alpha - \mathbf{1}^T\alpha$$
$$\text{s.t.} \quad \mathbf{y}^T\alpha = 0$$
$$0 \leq \alpha_n \leq C$$



Soft-Margin Dual SVM有两个应用非常广泛的工具包，分别是Libsvm和Liblinear。Libsvm和Liblinear都是国立台湾大学的Chih-Jen Lin博士开发的，Chih-Jen Lin的个人网站为：Welcome to Chih-Jen Lin's Home Page

下面我们再来回顾一下Soft-Margin SVM的主要内容。我们的出发点是用ξ_n来表示margin violation，即犯错值的大小，没有犯错对应的$\xi_n = 0$。然后将有条件问题转化为对偶dual形式，使用QP来得到最佳化的解。

从另外一个角度来看，ξ_n描述的是点(x_n, y_n)距离$y_n(w^T z_n + b) = 1$的边界有多远。第一种情况是violating margin，即不满足$y_n(w^T z_n + b) \geq 1$。那么ξ_n可表示为：$\xi_n = 1 - y_n(w^T z_n + b) > 0$。第二种情况是not violating margin，即点(x_n, y_n)在边界之外，满足$y_n(w^T z_n + b) \geq 1$的条件，此时$\xi_n = 0$。我们可以将两种情况整合到一个表达式中，对任意点：

$$\xi_n = max(1 - y_n(w^T z_n + b), 0)$$

上式表明，如果有voilating margin，则$1 - y_n(w^T z_n + b) > 0$，$\xi_n = 1 - y_n(w^T z_n + b)$；如果not violating margin，则$1 - y_n(w^T z_n + b) < 0$，$\xi_n = 0$。整合之后，我们可以把Soft-Margin SVM的最小化问题写成如下形式：

$$\frac{1}{2} w^T w + C \sum_{n=1}^{N} max(1 - y_n(w^T z_n + b), 0)$$

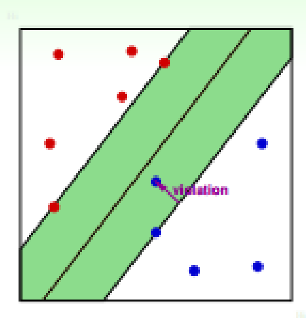
经过这种转换之后，表征犯错误值大小的变量ξ_n就被消去了，转而由一个max操作代替。



- record '**margin violation**' by ξ_n
- penalize with **margin violation**

$$\min_{b,\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C \cdot \sum_{n=1}^{N} \xi_n$$

$$\text{s.t.} \quad y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1 - \xi_n \text{ and } \xi_n \geq 0 \text{ for all } n$$



on any $(b, \mathbf{w})$, ξ_n = **margin violation** $= \max(1 - y_n(\mathbf{w}^T\mathbf{z}_n + b), 0)$

- $(\mathbf{x}_n, y_n)$ violating margin: $\xi_n = 1 - y_n(\mathbf{w}^T\mathbf{z}_n + b)$
- $(\mathbf{x}_n, y_n)$ not violating margin: $\xi_n = 0$

'unconstrained' form of soft-margin SVM:

$$\min_{b,\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N} \max(1 - y_n(\mathbf{w}^T\mathbf{z}_n + b), 0)$$

为什么要将把Soft-Margin SVM转换为这种unconstrained form呢？我们再来看一下转换后的形式，其中包含两项，第一项是w的内积，第二项关于y和w，b，z的表达式，似乎有点像一种错误估计$\hat{err}$，则类似这样的形式：

$$min\ \frac{1}{2}w^Tw + C\sum \hat{err}$$

看到这样的形式我们应该很熟悉，因为之前介绍的L2 Regularization中最优化问题的表达式跟这个是类似的：

$$min\ \frac{\lambda}{N}w^Tw + \frac{1}{N}\sum err$$

$$\min_{b,\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N} \max(1 - y_n(\mathbf{w}^T\mathbf{z}_n + b), 0)$$

familiar? :-)

$$\min \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum \widehat{\text{err}}$$

just L2 regularization

$$\min \quad \frac{\lambda}{N}\mathbf{w}^T\mathbf{w} + \frac{1}{N}\sum \text{err}$$

with shorter $\mathbf{w}$, another parameter, and special err



这里提一下，既然unconstrained form SVM与L2 Regularization的形式是一致的，而且L2 Regularization的解法我们之前也介绍过，那么为什么不直接利用这种方法来解决unconstrained form SVM的问题呢？有两个原因。一个是这种无条件的最优化问题无法通过QP解决，即对偶推导和kernel都无法使用；另一个是这种形式中包含的max()项可能造成函数并不是处处可导，这种情况难以用微分方法解决。

我们在第一节课中就介绍过Hard-Margin SVM与Regularization Model是有关系的。Regularization的目标是最小化E_{in}，条件是$w^Tw \leq C$，而Hard-Margin SVM的目标是最小化w^Tw，条件是$E_{in}=0$，即它们的最小化目标和限制条件是相互对调的。对于L2 Regularization来说，条件和最优化问题结合起来，整体形式写成：

$$\frac{\lambda}{N}w^Tw+E_{in}$$

而对于Soft-Margin SVM来说，条件和最优化问题结合起来，整体形式写成：

$$\frac{1}{2}w^Tw+CN\hat{E_{in}}$$

	minimize	constraint
regularization by constraint	E_{in}	$\mathbf{w}^T\mathbf{w} \leq C$
hard-margin SVM	$\mathbf{w}^T\mathbf{w}$	$E_{in}=0$ [and more]
L2 regularization	$\frac{\lambda}{N}\mathbf{w}^T\mathbf{w}+E_{in}$	
soft-margin SVM	$\frac{1}{2}\mathbf{w}^T\mathbf{w}+CN\widehat{E_{in}}$	

通过对比，我们发现L2 Regularization和Soft-Margin SVM的形式是相同的，两个式子分别包含了参数λ和C。Soft-Margin SVM中的large margin对应着L2 Regularization中的short w，也就是都让hyperplanes更简单一些。我们使用特别的$\hat{err}$来代表可以容忍犯错误的程度，即soft margin。L2 Regularization中的λ和Soft-Margin SVM中的C也是相互对应的，λ越大，w会越小，Regularization的程度就越大；C越小，$\hat{E_{in}}$会越大，相应的margin就越大。所以说增大C，或者减小λ，效果是一致的，Large-Margin等同于Regularization，都起到了防止过拟合的作用。

large margin $\Longleftrightarrow$ fewer hyperplanes $\Longleftrightarrow$ L2 regularization of short **w**

soft margin $\Longleftrightarrow$ special $\widehat{\text{err}}$

larger C or C $\Longleftrightarrow$ smaller λ $\Longleftrightarrow$ less regularization



建立了Regularization和Soft-Margin SVM的关系，接下来我们将尝试看看是否能把SVM作为一个regularized的模型进行扩展，来解决其它一些问题。

SVM versus Logistic Regression

上一小节，我们已经把Soft-Margin SVM转换成无条件的形式：

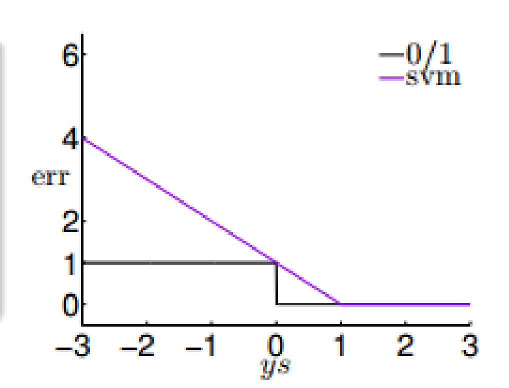
$$\min_{b,\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\max(1 - y_n(\mathbf{w}^T\mathbf{z}_n + b), 0)$$

上式中第二项的$max(1-y_n(w^Tz_n+b),0)$倍设置为$e\hat{r}r$。下面我们来看看$e\hat{r}r$与之前再二元分类中介绍过的$err_{0/1}$有什么关系。

对于$err_{0/1}$，它的linear score $s=w^Tz_n+b$，当$ys\geq 0$时，$err_{0/1}=0$；当$ys<0$时，$err_{0/1}=1$，呈阶梯状，如下图所示。而对于$e\hat{r}r$，当$ys\geq 0$时，$err_{0/1}=0$；当$ys<0$时，$err_{0/1}=1-ys$，呈折线状，如下图所示，通常把$e\hat{r}r_{svm}$称为hinge error measure。比较两条error曲线，我们发现$e\hat{r}r_{svm}$始终在$err_{0/1}$的上面，则$e\hat{r}r_{svm}$可作为$err_{0/1}$的上界。所以，可以使用$e\hat{r}r_{svm}$来代替$err_{0/1}$，解决二元线性分类问题，而且$e\hat{r}r_{svm}$是一个凸函数，使它在最佳化问题中有更好的性质。

linear score $s = \mathbf{w}^T\mathbf{z}_n + b$

- $\text{err}_{0/1}(s, y) = [\![ys \leq 0]\!]$
- $\widehat{\text{err}}_{\text{SVM}}(s, y) = \max(1 - ys, 0)$: upper bound of $\text{err}_{0/1}$ —often called **hinge error measure**



$\widehat{\text{err}}_{\text{SVM}}$: **algorithmic error measure** by **convex upper bound** of $\text{err}_{0/1}$

紧接着，我们再来看一下logistic regression中的error function。逻辑回归中，$err_{sce}=log_2(1+exp(-ys))$，当ys=0时，$err_{sce}=1$。它的err曲线如下所示。



linear score $s = \mathbf{w}^T\mathbf{z}_n + b$

- $\text{err}_{0/1}(s, y) = [\![ys \le 0]\!]$
- $\widehat{\text{err}}_{\text{SVM}}(s, y) = \max(1 - ys, 0)$: upper bound of $\text{err}_{0/1}$
- $\text{err}_{\text{SCE}}(s, y) = \log_2(1 + \exp(-ys))$: another upper bound of $\text{err}_{0/1}$ used in **logistic regression**

$-\infty$	$\longleftarrow$	ys	$\longrightarrow$	$+\infty$
$\approx -ys$		$\widehat{\text{err}}_{\text{SVM}}(s, y)$		$= 0$
$\approx -ys$		$(\ln 2) \cdot \text{err}_{\text{SCE}}(s, y)$		≈ 0

很明显，err_{sce}也是$err_{0/1}$的上界，而err_{sce}与$e\hat{r}r_{svm}$也是比较相近的。因为当ys趋向正无穷大的时候，err_{sce}和$e\hat{r}r_{svm}$都趋向于零；当ys趋向负无穷大的时候，err_{sce}和$e\hat{r}r_{svm}$都趋向于正无穷大。正因为二者的这种相似性，我们可以把SVM看成是L2-regularized logistic regression。

总结一下，我们已经介绍过几种Binary Classification的Linear Models，包括PLA，Logistic Regression和Soft-Margin SVM。PLA是相对简单的一个模型，对应的是$err_{0/1}$，通过不断修正错误的点来获得最佳分类线。它的优点是简单快速，缺点是只对线性可分的情况有用，线性不可分的情况需要用到pocket算法。Logistic Regression对应的是err_{sce}，通常使用GD/SGD算法求解最佳分类线。它的优点是凸函数err_{sce}便于最优化求解，而且有regularization作为避免过拟合的保证；缺点是err_{sce}作为$err_{0/1}$的上界，当ys很小（负值）时，上界变得更宽松，不利于最优化求解。Soft-Margin SVM对应的是$e\hat{r}r_{svm}$，通常使用QP求解最佳分类线。它的优点和Logistic Regression一样，凸优化问题计算简单而且分类线比较“粗壮”一些；缺点也和Logistic Regression一样，当ys很小（负值）时，上界变得过于宽松。其实，Logistic Regression和Soft-Margin SVM都是在最佳化$err_{0/1}$的上界而已。



PLA	soft-margin SVM	regularized logistic regression for classification
minimize $\text{err}_{0/1}$ specially • pros: **efficient if lin. separable** • cons: works only if lin. separable, otherwise needing **pocket**	minimize regularized $\widehat{\text{err}}_{\text{SVM}}$ by QP • pros: '**easy' optimization** & theoretical guarantee • cons: loose bound of $\text{err}_{0/1}$ for very negative *ys*	minimize regularized err_{SCE} by GD/SGD/... • pros: '**easy' optimization** & regularization guard • cons: loose bound of $\text{err}_{0/1}$ for very negative *ys*

至此，可以看出，求解regularized logistic regression的问题等同于求解soft-margin SVM的问题。反过来，如果我们求解了一个soft-margin SVM的问题，那这个解能否直接为regularized logistic regression所用？来预测结果是正类的几率是多少，就像regularized logistic regression做的一样。我们下一小节将来解答这个问题。

SVM for Soft Binary Classification

接下来，我们探讨如何将SVM的结果应用在Soft Binary Classification中，得到是正类的概率值。

第一种简单的方法是先得到SVM的解(b_{svm}, w_{svm})，然后直接代入到logistic regression中，得到$g(x) = \theta(w_{svm}^T x + b_{svm})$。这种方法直接使用了SVM和logistic regression的相似性，一般情况下表现还不错。但是，这种形式过于简单，与logistic regression的关联不大，没有使用到logistic regression中好的性质和方法。

第二种简单的方法是同样先得到SVM的解(b_{svm}, w_{svm})，然后把(b_{svm}, w_{svm})作为logistic regression的初始值，再进行迭代训练修正，速度比较快，最后，将得到的b和w代入到g(x)中。这种做法有点显得多此一举，因为并没有比直接使用logistic regression快捷多少。



Naïve Idea 1

1. run SVM and get $(b_{\text{SVM}}, \mathbf{w}_{\text{SVM}})$
2. return $g(\mathbf{x}) = \theta(\mathbf{w}_{\text{SVM}}^T \mathbf{x} + b_{\text{SVM}})$

- 'direct' use of similarity —works reasonably well
- **no LogReg flavor**

Naïve Idea 2

1. run SVM and get $(b_{\text{SVM}}, \mathbf{w}_{\text{SVM}})$
2. run LogReg with $(b_{\text{SVM}}, \mathbf{w}_{\text{SVM}})$ as $\mathbf{w}_0$
3. return LogReg solution as $g(\mathbf{x})$

- not really 'easier' than original LogReg
- **SVM flavor (kernel?) lost**

这两种方法都没有融合SVM和logistic regression各自的优势，下面构造一个模型，融合了二者的优势。构造的模型g(x)表达式为：

$$g(x) = \theta(A \cdot (w_{svm}^T \Phi(x) + b_{svm}) + B)$$

与上述第一种简单方法不同，我们额外增加了放缩因子A和平移因子B。首先利用SVM的解(b_{svm}, w_{svm})来构造这个模型，放缩因子A和平移因子B是待定系数。然后再用通用的logistic regression优化算法，通过迭代优化，得到最终的A和B。一般来说，如果(b_{svm}, w_{svm})较为合理的话，满足A>0且$B \approx 0$。

$$g(\mathbf{x}) = \theta(A \cdot (\mathbf{w}_{\text{SVM}}^T \mathbf{\Phi}(\mathbf{x}) + b_{\text{SVM}}) + B)$$

- **SVM flavor**: fix hyperplane direction by $\mathbf{w}_{\text{SVM}}$—**kernel** applies
- **LogReg flavor**: fine-tune hyperplane to match maximum likelihood by scaling (A) and shifting (B)
  - often $A > 0$ if $\mathbf{w}_{\text{SVM}}$ reasonably good
  - often $B \approx 0$ if b_{SVM} reasonably good

那么，新的logistic regression表达式为：

new LogReg Problem:

$$\min_{A,B} \quad \frac{1}{N}\sum_{n=1}^{N} \log\left(1 + \exp\left(-y_n\left(A \cdot \underbrace{(\mathbf{w}_{\text{SVM}}^T \mathbf{\Phi}(\mathbf{x}_n) + b_{\text{SVM}})}_{\Phi_{\text{SVM}}(\mathbf{x}_n)} + B\right)\right)\right)$$



这个表达式看上去很复杂，其实其中的(b_{svm}, w_{svm})已经在SVM中解出来了，实际上的未知参数只有A和B两个。归纳一下，这种Probabilistic SVM的做法分为三个步骤：

Platt's Model of Probabilistic SVM for Soft Binary Classification

1. run **SVM** on $\mathcal{D}$ to get $(b_{\text{SVM}}, \mathbf{w}_{\text{SVM}})$ [or the equivalent α], and transform $\mathcal{D}$ to $\mathbf{z}'_n = \mathbf{w}_{\text{SVM}}^T \mathbf{\Phi}(\mathbf{x}_n) + b_{\text{SVM}}$
   —actual model performs this step in a more complicated manner
2. run **LogReg** on $\{(\mathbf{z}'_n, y_n)\}_{n=1}^N$ to get (A, B)
   —actual model adds some special regularization here
3. return $g(\mathbf{x}) = \theta(A \cdot (\mathbf{w}_{\text{SVM}}^T \mathbf{\Phi}(\mathbf{x}) + b_{\text{SVM}}) + B)$

这种soft binary classifier方法得到的结果跟直接使用SVM classifier得到的结果可能不一样，这是因为我们引入了系数A和B。一般来说，soft binary classifier效果更好。至于logistic regression的解法，可以选择GD、SGD等等。

Kernel Logistic Regression

上一小节我们介绍的是通过kernel SVM在z空间中求得logistic regression的近似解。如果我们希望直接在z空间中直接求解logistic regression，通过引入kernel，来解决最优化问题，又该怎么做呢？SVM中使用kernel，转化为QP问题，进行求解，但是logistic regression却不是个QP问题，看似好像没有办法利用kernel来解决。

我们先来看看之前介绍的kernel trick为什么会work，kernel trick就是把z空间的内积转换到x空间中比较容易计算的函数。如果w可以表示为z的线性组合，即$w_* = \sum_{n=1}^N \beta_n z_n$的形式，那么乘积项$w_*^T z = \sum_{n=1}^N \beta_n z_n^T z = \sum_{n=1}^N \beta_n K(x_n, x)$，即其中包含了z的内积。也就是w可以表示为z的线性组合是kernel trick可以work的关键。

我们之前介绍过SVM、PLA包扩logistic regression都可以表示成z的线性组合，这也提供了一种可能，就是将kernel应用到这些问题中去，简化z空间的计算难度。



SVM	PLA	LogReg by SGD
$\mathbf{w}_{\text{SVM}} = \sum_{n=1}^{N} (\alpha_n y_n) \mathbf{z}_n$	$\mathbf{w}_{\text{PLA}} = \sum_{n=1}^{N} (\alpha_n y_n) \mathbf{z}_n$	$\mathbf{w}_{\text{LOGREG}} = \sum_{n=1}^{N} (\alpha_n y_n) \mathbf{z}_n$
α_n from **dual solutions**	α_n by **# mistake corrections**	α_n by **total SGD moves**

有这样一个理论，对于L2-regularized linear model，如果它的最小化问题形式为如下的话，那么最优解$w_* = \sum_{n=1}^{N} \beta_n z_n$。

claim: for any L2-regularized linear model

$$\min_{\mathbf{w}} \quad \frac{\lambda}{N}\mathbf{w}^T\mathbf{w} + \frac{1}{N}\sum_{n=1}^{N} \text{err}(y_n, \mathbf{w}^T\mathbf{z}_n)$$

optimal $\mathbf{w}_* = \sum_{n=1}^{N} \beta_n \mathbf{z}_n$.

下面给出简单的证明，假如最优解$w_* = w_{||} + w_{\perp}$。其中，$w_{||}$和$w_{\perp}$分别是平行z空间和垂直z空间的部分。我们需要证明的是$w_{\perp} = 0$。利用反证法，假如$w_{\perp} \neq 0$，考虑w_*与$w_{||}$的比较。第一步先比较最小化问题的第二项：$err(y, w_*^T z_n) = err(y_n, (w_{||} + w_{\perp})^T z_n = err(y_n, w_{||}^T z_n)$，即第二项是相等的。然后第二步比较第一项：$w_*^T w_* = w_{||}^T w_{||} + 2w_{||}^T w_{\perp} + w_{\perp}^T w_{\perp} > w_{||}^T w_{||}$，即$w_*$对应的L2-regularized linear model值要比$w_{||}$大，这就说明w_*并不是最优解，从而证明$w_{\perp}$必然等于零，即$w_* = \sum_{n=1}^{N} \beta_n z_n$一定成立，$w_*$一定可以写成z的线性组合形式。

- let optimal $\mathbf{w}_* = \mathbf{w}_{||} + \mathbf{w}_{\perp}$, where $\mathbf{w}_{||} \in \text{span}(\mathbf{z}_n)$ & $\mathbf{w}_{\perp} \perp \text{span}(\mathbf{z}_n)$
  —want $\mathbf{w}_{\perp} = \mathbf{0}$
- what if **not?** Consider $\mathbf{w}_{||}$
  - of same err as $\mathbf{w}_*$: $\text{err}(y_n, \mathbf{w}_*^T\mathbf{z}_n) = \text{err}(y_n, (\mathbf{w}_{||} + \mathbf{w}_{\perp})^T\mathbf{z}_n)$
  - of smaller regularizer as $\mathbf{w}_*$:
    $\mathbf{w}_*^T\mathbf{w}_* = \mathbf{w}_{||}^T\mathbf{w}_{||} + 2\mathbf{w}_{||}^T\mathbf{w}_{\perp} + \mathbf{w}_{\perp}^T\mathbf{w}_{\perp} \; > \; \mathbf{w}_{||}^T\mathbf{w}_{||}$

  —$\mathbf{w}_{||}$ **'more optimal'** than $\mathbf{w}_*$ **(contradiction!)**

经过证明和分析，我们得到了结论是任何L2-regularized linear model都可以使用



kernel来解决。

现在，我们来看看如何把kernel应用在L2-regularized logistic regression上。上面我们已经证明了w_*一定可以写成z的线性组合形式，即$w_* = \sum_{n=1}^N \beta_n z_n$。那么我们就无需一定求出$w_*$，而只要求出其中的$\beta_n$就行了。怎么求呢？直接将$w_* = \sum_{n=1}^N \beta_n z_n$代入到L2-regularized logistic regression最小化问题中，得到：

solving L2-regularized logistic regression

$$\min_{\mathbf{w}} \quad \frac{\lambda}{N}\mathbf{w}^T\mathbf{w} + \frac{1}{N}\sum_{n=1}^{N}\log\left(1+\exp\left(-y_n\mathbf{w}^T\mathbf{z}_n\right)\right)$$

yields optimal solution $\mathbf{w}_* = \sum_{n=1}^N \beta_n \mathbf{z}_n$

with out loss of generality, can solve for optimal β instead of $\mathbf{w}$

$$\min_{\beta} \frac{\lambda}{N}\sum_{n=1}^{N}\sum_{m=1}^{N}\beta_n\beta_m K(\mathbf{x}_n,\mathbf{x}_m) + \frac{1}{N}\sum_{n=1}^{N}\log\left(1+\exp\left(-y_n\sum_{m=1}^{N}\beta_m K(\mathbf{x}_m,\mathbf{x}_n)\right)\right)$$

—how? GD/SGD/... for unconstrained optimization

上式中，所有的w项都换成β_n来表示了，变成了没有条件限制的最优化问题。我们把这种问题称为kernel logistic regression，即引入kernel，将求w的问题转换为求β_n的问题。

从另外一个角度来看Kernel Logistic Regression（KLR）：

$$\min_{\beta} \frac{\lambda}{N}\sum_{n=1}^{N}\sum_{m=1}^{N}\beta_n\beta_m K(\mathbf{x}_n,\mathbf{x}_m) + \frac{1}{N}\sum_{n=1}^{N}\log\left(1+\exp\left(-y_n\sum_{m=1}^{N}\beta_m K(\mathbf{x}_m,\mathbf{x}_n)\right)\right)$$

上式中log项里的$\sum_{m=1}^N \beta_m K(x_m, x_n)$可以看成是变量$\beta$和$K(x_m, x_n)$的内积。上式第一项中的$\sum_{n=1}^N \sum_{m=1}^N \beta_n\beta_m K(x_n, x_m)$可以看成是关于$\beta$的正则化项$\beta^T K\beta$。所以，KLR是$\beta$的线性组合，其中包含了kernel内积项和kernel regularizer。这与SVM是相似的形式。

但值得一提的是，KLR中的β_n与SVM中的α_n是有区别的。SVM中的α_n大部分为零，



SV的个数通常是比较少的；而KLR中的β_n通常都是非零值。

总结

本节课主要介绍了Kernel Logistic Regression。首先把Soft-Margin SVM解释成Regularized Model，建立二者之间的联系，其实Soft-Margin SVM就是一个L2-regularization，对应着hinge error messure。然后利用它们之间的相似性，讨论了如何利用SVM的解来得到Soft Binary Classification。方法是先得到SVM的解，再在logistic regression中引入参数A和B，迭代训练，得到最佳解。最后介绍了Kernel Logistic Regression，证明L2-regularized logistic regression中，最佳解w_*一定可以写成z的线性组合形式，从而可以将kernel引入logistic regression中，使用kernel思想在z空间直接求解L2-regularized logistic regression问题。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程



林轩田《机器学习技法》课程笔记6 -- Support Vector Regression

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了Kernel Logistic Regression，讨论如何把SVM的技巧应用在soft-binary classification上。方法是使用2-level learning，先利用SVM得到参数b和w，然后再用通用的logistic regression优化算法，通过迭代优化，对参数b和w进行微调，得到最佳解。然后，也介绍了可以通过Representer Theorem，在z空间中，引入SVM的kernel技巧，直接对logistic regression进行求解。本节课将延伸上节课的内容，讨论如何将SVM的kernel技巧应用到regression问题上。

Kernel Ridge Regression

首先回顾一下上节课介绍的Representer Theorem，对于任何包含正则项的L2-regularized linear model，它的最佳化解w都可以写成是z的线性组合形式，因此，也就能引入kernel技巧，将模型kernelized化。

for any L2-regularized linear model

$$\min_{\mathbf{w}} \quad \frac{\lambda}{N}\mathbf{w}^T\mathbf{w} + \frac{1}{N}\sum_{n=1}^{N} \text{err}(y_n, \mathbf{w}^T\mathbf{z}_n)$$

optimal $\mathbf{w}_* = \sum_{n=1}^{N} \beta_n \mathbf{z}_n$.

—any L2-regularized linear model can be **kernelized**!

那么如何将regression模型变成kernel的形式呢？我们之前介绍的linear/ridge regression最常用的错误估计是squared error，即$err(y, w^Tz) = (y - w^Tz)^2$。这种形式对应的解是analytic solution，即可以使用线性最小二乘法，通过向量运算，直接得到最优化解。那么接下来我们就要研究如何将kernel引入到ridge regression中去，得到与之对应的analytic solution。

我们先把Kernel Ridge Regression问题写下来：



solving ridge regression $\min_{\mathbf{w}} \frac{\lambda}{N}\mathbf{w}^T\mathbf{w} + \frac{1}{N}\sum_{n=1}^{N}(y_n - \mathbf{w}^T\mathbf{z}_n)^2$

yields optimal solution $\mathbf{w}_* = \sum_{n=1}^{N}\beta_n\mathbf{z}_n$

其中，最佳解w_*必然是z的线性组合。那么我们就把$w_* = \sum_{n=1}^{N}\beta_n z_n$代入到ridge regression中，将z的内积用kernel替换，把求w_*的问题转化成求β_n的问题，得到：

with out loss of generality, can solve for optimal β instead of $\mathbf{w}$

$$\min_{\beta} \quad \frac{\lambda}{N}\underbrace{\sum_{n=1}^{N}\sum_{m=1}^{N}\beta_n\beta_m K(\mathbf{x}_n,\mathbf{x}_m)}_{\text{regularization of } \beta \text{ on } K\text{-based regularizer}} + \underbrace{\frac{1}{N}\sum_{n=1}^{N}\left(y_n - \sum_{m=1}^{N}\beta_m K(\mathbf{x}_n,\mathbf{x}_m)\right)^2}_{\text{linear regression of } \beta \text{ on } K\text{-based features}}$$

$$= \frac{\lambda}{N}\beta^T\mathrm{K}\beta + \frac{1}{N}\left(\beta^T\mathrm{K}^T\mathrm{K}\beta - 2\beta^T\mathrm{K}^T\mathbf{y} + \mathbf{y}^T\mathbf{y}\right)$$

ridge regression可以写成矩阵的形式，其中第一项可以看成是β_n的正则项，而第二项可以看成是β_n的error function。这样，我们的目的就是求解该式最小化对应的β_n值，这样就解决了kernel ridge regression问题。

求解β_n的问题可以写成如下形式：

$$E_{\text{aug}}(\beta) = \frac{\lambda}{N}\beta^T\mathrm{K}\beta + \frac{1}{N}\left(\beta^T\mathrm{K}^T\mathrm{K}\beta - 2\beta^T\mathrm{K}^T\mathbf{y} + \mathbf{y}^T\mathbf{y}\right)$$

$$\nabla E_{\text{aug}}(\beta) = \frac{2}{N}\left(\lambda\mathrm{K}^T\mathrm{I}\beta + \mathrm{K}^T\mathrm{K}\beta - \mathrm{K}^T\mathbf{y}\right) = \frac{2}{N}\mathrm{K}^T\left((\lambda\mathrm{I} + \mathrm{K})\beta - \mathbf{y}\right)$$

$E_{aug}(\beta)$是关于β的二次多项式，要对$E_{aug}(\beta)$求最小化解，这种凸二次最优化问题，只需要先计算其梯度，再令梯度为零即可。$\nabla E_{aug}(\beta)$已经在上式中写出来了，令其等于零，即可得到一种可能的β的解析解为：

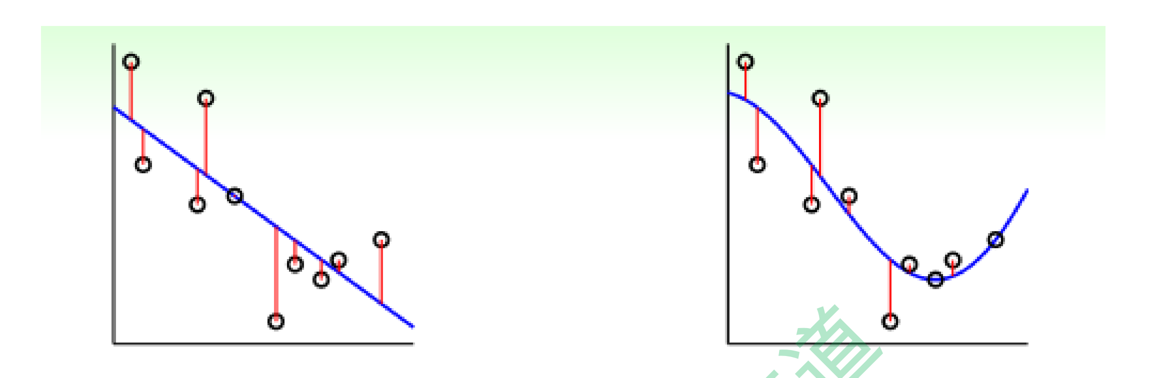
$$\beta = (\lambda I + K)^{-1}y$$

这里需要关心的问题是$(\lambda I + K)$的逆矩阵是否存在？答案是肯定的。因为我们之前介绍过，核函数K满足Mercer's condition，它是半正定的，而且$\lambda > 0$，所以$(\lambda I + K)$一定是可逆的。从计算的时间复杂上来说，由于$(\lambda I + K)$是NxN大小的，所以时间复杂度是$O(N^3)$。还有一点，$\nabla E_{aug}(\beta)$是由两项乘积构成的，另一项是



K，会不会出现K=0的情况呢？其实，由于核函数K表征的是z空间的内积，一般而言，除非两个向量互相垂直，内积才为零，否则，一般情况下K不等于零。这个原因也决定了$(\lambda I+K)$是dense matrix，即β的解大部分都是非零值。这个性质，我们之后还会说明。

所以说，我们可以通过kernel来解决non-linear regression的问题。下面比较一下linear ridge regression和kernel ridge regression的关系。



如上图所示，左边是linear ridge regression，是一条直线；右边是kernel ridge regression，是一条曲线。大致比较一下，右边的曲线拟合的效果更好一些。这两种regression有什么样的优点和缺点呢？对于linear ridge regression来说，它是线性模型，只能拟合直线；其次，它的训练复杂度是$O(d^3+d^2N)$，预测的复杂度是$O(d)$，如果N比d大很多时，这种模型就更有效率。而对于kernel ridge regression来说，它转换到z空间，使用kernel技巧，得到的是非线性模型，所以更加灵活；其次，它的训练复杂度是$O(N^3)$，预测的复杂度是$O(N)$，均只与N有关。当N很大的时候，计算量就很大，所以，kernel ridge regression适合N不是很大的场合。比较下来，可以说linear和kernel实际上是效率（efficiency）和灵活（flexibility）之间的权衡。

linear ridge regression	kernel ridge regression
$\mathbf{w} = (\lambda \mathrm{I}+\mathrm{X}^T\mathrm{X})^{-1}\mathrm{X}^T\mathbf{y}$	$\beta = (\lambda \mathrm{I}+\mathrm{K})^{-1}\mathbf{y}$
• more restricted	• more flexible with K
• $O(d^3+d^2N)$ training; $O(d)$ prediction —efficient when $N \gg d$	• $O(N^3)$ training; $O(N)$ prediction —hard for big data

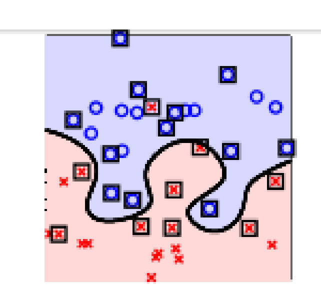
linear versus **kernel**:
trade-off between **efficiency** and **flexibility**

Support Vector Regression Primal

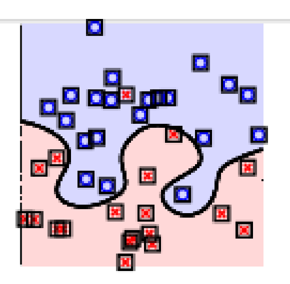


我们在机器学习基石课程中介绍过linear regression可以用来做classification，那么上一部分介绍的kernel ridge regression同样可以来做classification。我们把kernel ridge regression应用在classification上取个新的名字，叫做least-squares SVM（LSSVM）。

先来看一下对于某个问题，soft-margin Gaussian SVM和Gaussian LSSVM结果有哪些不一样的地方。



soft-margin Gaussian SVM



Gaussian LSSVM

如上图所示，如果只看分类边界的话，soft-margin Gaussian SVM和Gaussian LSSVM差别不是很大，即的到的分类线是几乎相同的。但是如果看Support Vector的话（图中方框标注的点），左边soft-margin Gaussian SVM的SV不多，而右边Gaussian LSSVM中基本上每个点都是SV。这是因为soft-margin Gaussian SVM中的α_n大部分是等于零，$\alpha_n > 0$的点只占少数，所以SV少。而对于LSSVM，我们上一部分介绍了β的解大部分都是非零值，所以对应的每个点基本上都是SV。SV太多会带来一个问题，就是做预测的矩$g(x)=\sum_{n=1}^N \beta_n K(x_n,x)$，如果$\beta_n$非零值较多，那么g的计算量也比较大，降低计算速度。基于这个原因，soft-margin Gaussian SVM更有优势。

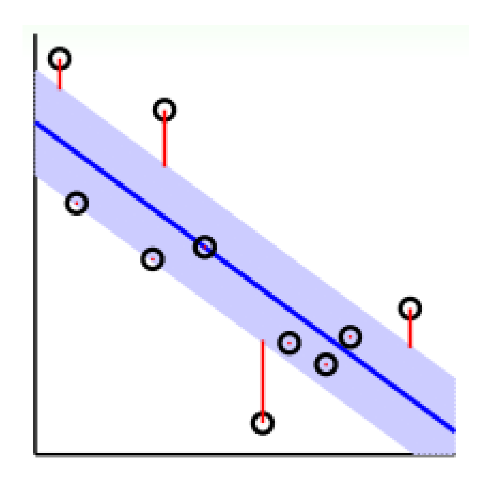
- LSSVM: similar boundary, **many more SVs**
  $\Longrightarrow$ slower prediction, **dense β (BIG g)**
- dense β: LSSVM, kernel LogReg;
  sparse α: standard SVM

那么，针对LSSVM中dense β的缺点，我们能不能使用一些方法来的得到sparse β，使得SV不会太多，从而得到和soft-margin SVM同样的分类效果呢？下面我们将尝试解决这个问题。

方法是引入一个叫做Tube Regression的做法，即在分类线上下分别划定一个区域（中立区），如果数据点分布在这个区域内，则不算分类错误，只有误分在中立区域



之外的地方才算error。



假定中立区的宽度为2ϵ，$\epsilon > 0$,那么error measure就可以写成：
$err(y,s) = max(0, |s-y| - \epsilon)$，对应上图中红色标注的距离。

error measure:

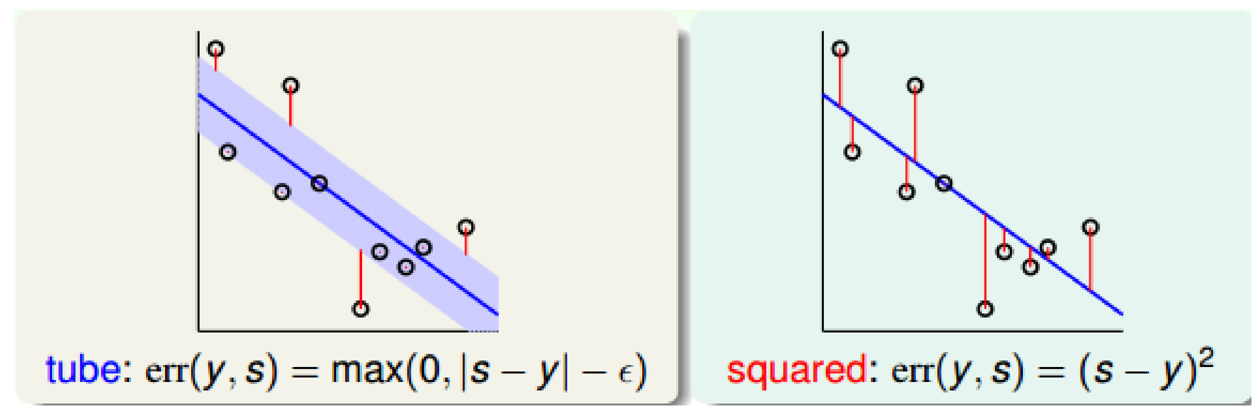
$$\text{err}(y,s) = \max(0, |s-y| - \epsilon)$$

- $|s-y| \le \epsilon$: 0
- $|s-y| > \epsilon$: $|s-y| - \epsilon$

—usually called ϵ**-insensitive error** with $\epsilon > 0$

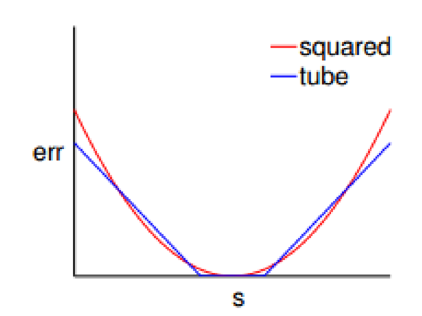
通常把这个error叫做ϵ-insensitive error，这种max的形式跟我们上节课中介绍的hinge error measure形式其实是类似的。所以，我们接下来要做的事情就是将L2-regularized tube regression做类似于soft-margin SVM的推导，从而得到sparse β。

首先，我们把tube regression中的error与squared error做个比较：



然后，将err(y,s)与s的关系曲线分别画出来：





tube $\approx$ squared when $|s - y|$ small
& **less affected by outliers**

上图中，红色的线表示squared error，蓝色的线表示tube error。我们发现，当|s-y|比较小即s比较接近y的时候，squared error与tube error是差不多大小的。而在|s-y|比较大的区域，squared error的增长幅度要比tube error大很多。error的增长幅度越大，表示越容易受到noise的影响，不利于最优化问题的求解。所以，从这个方面来看，tube regression的这种error function要更好一些。

现在，我们把L2-Regularized Tube Regression写下来：

$$\min_{\mathbf{w}} \quad \frac{\lambda}{N}\mathbf{w}^T\mathbf{w} + \frac{1}{N}\sum_{n=1}^{N} \max\left(0, |\mathbf{w}^T\mathbf{z}_n - y| - \epsilon\right)$$

这个最优化问题，由于其中包含max项，并不是处处可微分的，所以不适合用GD/SGD来求解。而且，虽然满足representer theorem，有可能通过引入kernel来求解，但是也并不能保证得到sparsity β。从另一方面考虑，我们可以把这个问题转换为带条件的QP问题，仿照dual SVM的推导方法，引入kernel，得到KKT条件，从而保证解β是sparse的。

Regularized Tube Regr.

$\min \frac{\lambda}{N}\mathbf{w}^T\mathbf{w} + \frac{1}{N}\sum$ tube violation

- unconstrained,
  but max **not differentiable**
- 'representer' to kernelize,
  but **no obvious sparsity**

standard SVM

$\min \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum$ margin vio.

- not differentiable,
  but **QP**
- dual to kernelize,
  KKT conditions $\Rightarrow$ **sparsity**

所以，我们就可以把L2-Regularized Tube Regression写成跟SVM类似的形式：



will mimic standard SVM derivation:

$$\min_{b,\mathbf{w}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N} \max\left(0, |\mathbf{w}^T\mathbf{z}_n + b - y_n| - \epsilon\right)$$

值得一提的是，系数λ和C是反比例相关的，λ越大对应C越小，λ越小对应C越大。而且该式也把w_0即b单独拿了出来，这跟我们之前推导SVM的解的方法是一致的。

现在我们已经有了Standard Support Vector Regression的初始形式，这还是不是一个标准的QP问题。我们继续对该表达式做一些转化和推导：

mimicking standard SVM

$$\min_{b,\mathbf{w},\xi} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N} \xi_n$$
$$s.t. \quad |\mathbf{w}^T\mathbf{z}_n + b - y_n| \le \epsilon + \xi_n$$
$$\xi_n \ge 0$$

making constraints linear

$$\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N} (\xi_n^{\vee} + \xi_n^{\wedge})$$
$$-\epsilon - \xi_n^{\vee} \le y_n - \mathbf{w}^T\mathbf{z}_n - b \le \epsilon + \xi_n^{\wedge}$$
$$\xi_n^{\vee} \ge 0, \xi_n^{\wedge} \ge 0$$

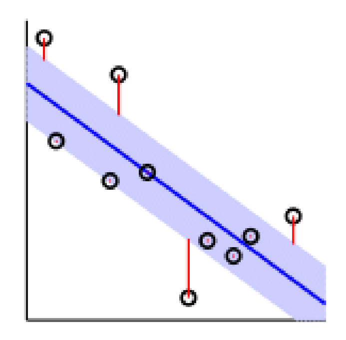
如上图右边所示，即为标准的QP问题，其中$\xi_n^{\vee}$和$\xi_n^{\wedge}$分别表示upper tube violations和lower tube violations。这种形式叫做Support Vector Regression（SVR） primal。

$$\min_{b,\mathbf{w},\xi^{\vee},\xi^{\wedge}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N} (\xi_n^{\vee} + \xi_n^{\wedge})$$
$$s.t. \quad -\epsilon - \xi_n^{\vee} \le y_n - \mathbf{w}^T\mathbf{z}_n - b \le \epsilon + \xi_n^{\wedge}$$
$$\xi_n^{\vee} \ge 0, \xi_n^{\wedge} \ge 0$$

SVR的标准QP形式包含几个重要的参数：C和ϵ。C表示的是regularization和tube violation之间的权衡。large C倾向于tube violation，small C则倾向于regularization。ϵ表征了tube的区域宽度，即对错误点的容忍程度。ϵ越大，则表示对错误的容忍度越大。ϵ是可设置的常数，是SVR问题中独有的，SVM中没有这个参数。另外，SVR的QP形式共有$\hat{d}+1+2N$个参数，2N+2N个条件。



- parameter C: trade-off of regularization & tube violation
- parameter ϵ: vertical tube width —one more parameter to choose!
- QP of $\tilde{d} + 1 + 2N$ variables, $2N + 2N$ constraints



Support Vector Regression Dual

现在我们已经得到了SVR的primal形式，接下来将推导SVR的Dual形式。首先，与SVM对偶形式一样，先令拉格朗日因子$\alpha^{\vee}$和$\alpha^{\wedge}$，分别是与$\xi_n^{\vee}$和$\xi_n^{\wedge}$不等式相对应。这里忽略了与$\xi_n^{\vee} \geq 0$和$\xi_n^{\wedge} \geq 0$对应的拉格朗日因子。

objective function		$\frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}(\xi_n^{\vee} + \xi_n^{\wedge})$
Lagrange multiplier $\alpha_n^{\wedge}$	for	$y_n - \mathbf{w}^T\mathbf{z}_n - b \leq \epsilon + \xi_n^{\wedge}$
Lagrange multiplier $\alpha_n^{\vee}$	for	$-\epsilon - \xi_n^{\vee} \leq y_n - \mathbf{w}^T\mathbf{z}_n - b$

然后，与SVM一样做同样的推导和化简，拉格朗日函数对相关参数偏微分为零，得到相应的KKT条件：

Some of the KKT Conditions

- $\frac{\partial \mathcal{L}}{\partial w_i} = 0$: $\mathbf{w} = \sum_{n=1}^{N} \underbrace{(\alpha_n^{\wedge} - \alpha_n^{\vee})}_{\beta_n} \mathbf{z}_n$; $\frac{\partial \mathcal{L}}{\partial b} = 0$: $\sum_{n=1}^{N}(\alpha_n^{\wedge} - \alpha_n^{\vee}) = 0$
- complementary slackness:

$$\alpha_n^{\wedge}(\epsilon + \xi_n^{\wedge} - y_n + \mathbf{w}^T\mathbf{z}_n + b) = 0$$
$$\alpha_n^{\vee}(\epsilon + \xi_n^{\vee} + y_n - \mathbf{w}^T\mathbf{z}_n - b) = 0$$

接下来，通过观察SVM primal与SVM dual的参数对应关系，直接从SVR primal推导出SVR dual的形式。（具体数学推导，此处忽略！）



$$
\begin{aligned}
\min \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}\xi_n \\
\text{s.t.} \quad & y_n(\mathbf{w}^T\mathbf{z}_n + b) \geq 1 - \xi_n \\
& \xi_n \geq 0
\end{aligned}
$$

$$
\begin{aligned}
\min \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{n=1}^{N}(\xi_n^{\wedge} + \xi_n^{\vee}) \\
\text{s.t.} \quad & 1(y_n - \mathbf{w}^T\mathbf{z}_n - b) \leq \epsilon + \xi_n^{\wedge} \\
& 1(\mathbf{w}^T\mathbf{z}_n + b - y_n) \leq \epsilon + \xi_n^{\vee} \\
& \xi_n^{\wedge} \geq 0, \xi_n^{\vee} \geq 0
\end{aligned}
$$

$$
\begin{aligned}
\min \quad & \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}\alpha_n\alpha_m y_n y_m K(\mathbf{x}_n, \mathbf{x}_m) \\
& -\sum_{n=1}^{N} 1 \cdot \alpha_n \\
\text{s.t.} \quad & \sum_{n=1}^{N} y_n\alpha_n = 0 \\
& 0 \leq \alpha_n \leq C
\end{aligned}
$$

$$
\begin{aligned}
\min \quad & \frac{1}{2}\sum_{n=1}^{N}\sum_{m=1}^{N}(\alpha_n^{\wedge} - \alpha_n^{\vee})(\alpha_m^{\wedge} - \alpha_m^{\vee})k_{n,m} \\
& +\sum_{n=1}^{N}\left((\epsilon - y_n)\cdot\alpha_n^{\wedge} + (\epsilon + y_n)\cdot\alpha_n^{\vee}\right) \\
\text{s.t.} \quad & \sum_{n=1}^{N} 1 \cdot (\alpha_n^{\wedge} - \alpha_n^{\vee}) = 0 \\
& 0 \leq \alpha_n^{\wedge} \leq C, 0 \leq \alpha_n^{\vee} \leq C
\end{aligned}
$$

最后，我们就要来讨论一下SVR的解是否真的是sparse的。前面已经推导了SVR dual形式下推导的解w为：

$$w = \sum_{n=1}^{N}(\alpha_n^{\wedge} - \alpha_n^{\vee})z_n$$

相应的complementary slackness为：

$$
\begin{aligned}
\alpha_n^{\wedge}(\epsilon + \xi_n^{\wedge} - y_n + \mathbf{w}^T\mathbf{z}_n + b) &= 0 \\
\alpha_n^{\vee}(\epsilon + \xi_n^{\vee} + y_n - \mathbf{w}^T\mathbf{z}_n - b) &= 0
\end{aligned}
$$

对于分布在tube中心区域内的点，满足$|w^T z_n + b - y_n| < \epsilon$，此时忽略错误，$\xi_n^{\vee}$和$\xi_n^{\wedge}$都等于零。则complementary slackness两个等式的第二项均不为零，必然得到$\alpha_n^{\wedge} = 0$和$\alpha_n^{\vee} = 0$，即$\beta_n = \alpha_n^{\wedge} - \alpha_n^{\vee} = 0$。

所以，对于分布在tube内的点，得到的解$\beta_n = 0$，是sparse的。而分布在tube之外的点，$\beta_n \neq 0$。至此，我们就得到了SVR的sparse解。

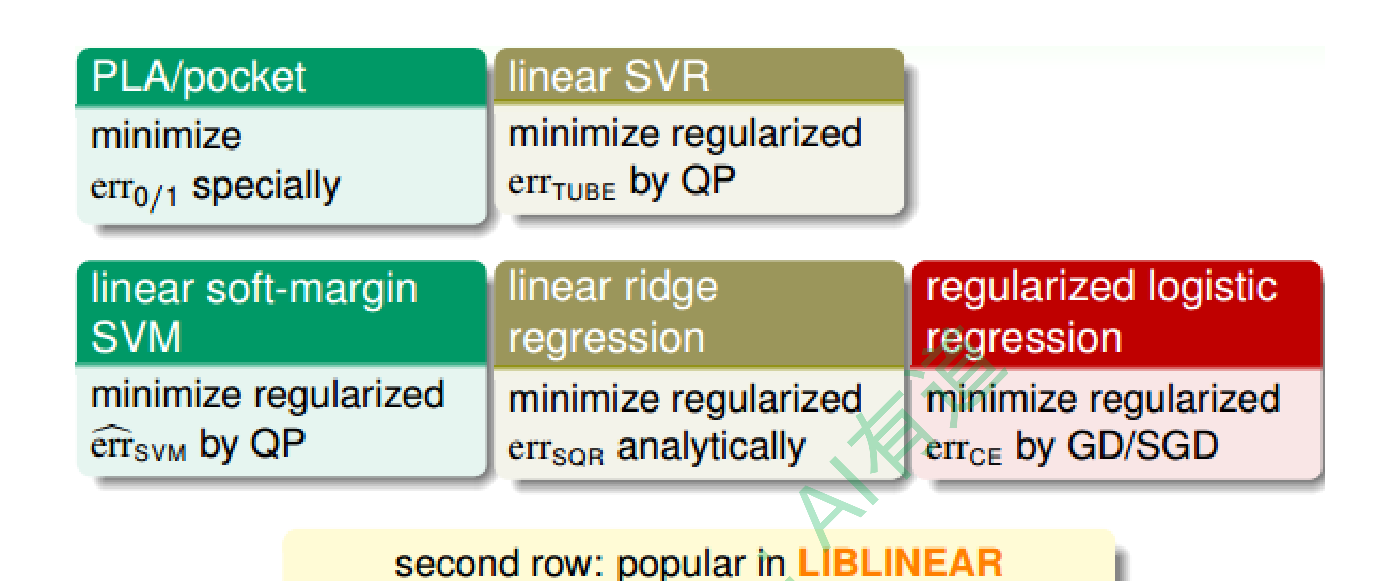
Summary of Kernel Models

这部分将对我们介绍过的所有的kernel模型做个概括和总结。我们总共介绍过三种线性模型，分别是PLA/pocket，regularized logistic regression和linear ridge regression。这三种模型都可以使用国立台湾大学的Chih-Jen Lin博士开发的Liblinea



库函数来解决。

另外，我们介绍了linear soft-margin SVM，其中的error function是$\hat{err}_{svm}$，可以通过标准的QP问题来求解。linear soft-margin SVM和PLA/pocket一样都是解决同样的问题。然后，还介绍了linear SVR问题，它与linear ridge regression一样都是解决同样的问题，从SVM的角度，使用err_{tube}，转换为QP问题进行求解，这也是我们本节课的主要内容。



上图中相应的模型也可以转化为dual形式，引入kernel，整体的框图如下：

PLA/pocket
linear SVR
linear soft-margin SVM
linear ridge regression
regularized logistic regression
kernel ridge regression
kernelized linear ridge regression
kernel logistic regression
kernelized regularized logistic regression
SVM
minimize SVM dual by QP
SVR
minimize SVR dual by QP
probabilistic SVM
run SVM-transformed logistic regression
fourth row: popular in LIBSVM



其中SVM，SVR和probabilistic SVM都可以使用国立台湾大学的Chih-Jen Lin博士开发的LLibsvm库函数来解决。通常来说，这些模型中SVR和probabilistic SVM最为常用。

总结

本节课主要介绍了SVR，我们先通过representer theorem理论，将ridge regression转化为kernel的形式，即kernel ridge regression，并推导了SVR的解。但是得到的解是dense的，大部分为非零值。所以，我们定义新的tube regression，使用SVM的推导方法，来最小化regularized tube errors，转化为对偶形式，得到了sparse的解。最后，我们对介绍过的所有kernel模型做个总结，简单概述了各自的特点。在实际应用中，我们要根据不同的问题进行合适的模型选择。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程

林轩田《机器学习技法》课程笔记7 -- Blending and Bagging

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了Support Vector Regression，将kernel model引入到regression中。首先，通过将ridge regression和representer theorem结合起来，得到kernel ridge regression。但是其解是dense的，即不部分不为零。为了得到sparse解，我们将regularized tube error和Lagrange dual结合起来，利用SVM dual的推导方法，得到support vector regression的sparse解。本系列1-6节课主要介绍Kernel Models及其应用，从本节课开始，讲介绍Aggregation Models，即如何将不同的hypothesis和features结合起来，让模型更好。本节课将介绍其中的两个方法，一个是Blending，一个是Bagging。

Motivation of Aggregation

首先举个例子来说明为什么要使用Aggregation。假如你有T个朋友，每个朋友向你预测推荐明天某支股票会涨还是会跌，对应的建议分别是$g_1, g_2, \cdots, g_T$，那么你该选择哪个朋友的建议呢？即最终选择对股票预测的$g_t(x)$是什么样的？

第一种方法是从T个朋友中选择一个最受信任，对股票预测能力最强的人，直接听从他的建议就好。这是一种普遍的做法，对应的就是validation思想，即选择犯错误最小的模型。第二种方法，如果每个朋友在股票预测方面都是比较厉害的，都有各自的专长，那么就同时考虑T个朋友的建议，将所有结果做个投票，一人一票，最终决定出对该支股票的预测。这种方法对应的是uniformly思想。第三种方法，如果每个朋友水平不一，有的比较厉害，投票比重应该更大一些，有的比较差，投票比重应该更小一些。那么，仍然对T个朋友进行投票，只是每个人的投票权重不同。这种方法对应的是non-uniformly的思想。第四种方法与第三种方法类似，但是权重不是固定的，根据不同的条件，给予不同的权重。比如如果是传统行业的股票，那么给这方面比较厉害的朋友较高的投票权重，如果是服务行业，那么就给这方面比较厉害的朋友较高的投票权重。以上所述的这四种方法都是将不同人不同意见融合起来的方式，接下来我们就要讨论如何将这些做法对应到机器学习中去。Aggregation的思想与这个例子是类似的，即把多个hypothesis结合起来，得到更好的预测效果。

You can . . .

- **select** the most trust-worthy friend from their **usual performance**
  —**validation!**
- **mix** the predictions from all your friends **uniformly**
  —let them **vote!**
- **mix** the predictions from all your friends **non-uniformly**
  —let them vote, but **give some more ballots**
- **combine** the predictions **conditionally**
  —if **[*t* satisfies some condition]** give some ballots to friend *t*
- ...

将刚刚举的例子的各种方法用数学化的语言和机器学习符号归纳表示出来，其中G(x)表示最终选择的模型。

第一种方法对应的模型：

$$G(x) = g_{t_*}(x)\ with\ t_* = argmin_{t\in 1,2,\cdots,T}\ E_{val}(g_t^-)$$

第二种方法对应的模型：

$$G(x) = sign(\sum_{t=1}^{T} 1 \cdot g_t(x))$$

第三种方法对应的模型：

$$G(x) = sign(\sum_{t=1}^{T} \alpha_t \cdot g_t(x))\ with\ \alpha_t \geq 0$$

第四种方法对应的模型：

$$G(x) = sign(\sum_{t=1}^{T} q_t(x) \cdot g_t(x))\ with\ q_t(x) \geq 0$$



- **select** the most trust-worthy friend from their **usual performance**

$$G(\mathbf{x}) = g_{t_*}(\mathbf{x}) \text{ with } t_* = \text{argmin}_{t\in\{1,2,\cdots,T\}} E_{\text{val}}(g_t^-)$$

- **mix** the predictions from all your friends **uniformly**

$$G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} 1 \cdot g_t(\mathbf{x})\right)$$

- **mix** the predictions from all your friends **non-uniformly**

$$G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \cdot g_t(\mathbf{x})\right) \text{ with } \alpha_t \geq 0$$

  - include **select**: $\alpha_t = [\![E_{\text{val}}(g_t^-) \text{ smallest}]\!]$
  - include **uniformly**: $\alpha_t = 1$

- **combine** the predictions **conditionally**

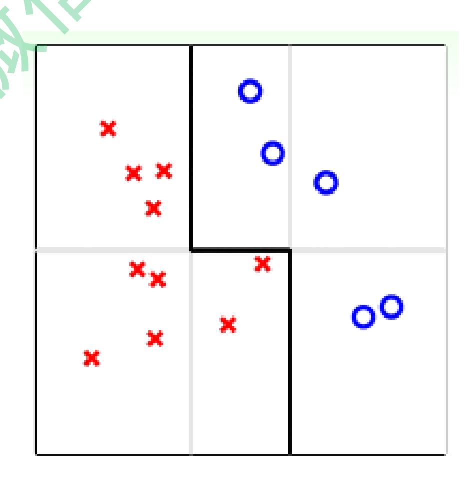
$$G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} q_t(\mathbf{x}) \cdot g_t(\mathbf{x})\right) \text{ with } q_t(\mathbf{x}) \geq 0$$

  - include **non-uniformly**: $q_t(\mathbf{x}) = \alpha_t$

注意这里提到的第一种方法是通过验证集来选择最佳模型，不能使用$E_{in}(g_t)$来代替$E_{val}(g_t^-)$。经过Validation，选择最小的E_{val}，保证E_{out}最小，从而将对应的模型作为最佳的选择。

但是第一种方法只是从众多可能的hypothesis中选择最好的模型，并不能发挥集体的智慧。而Aggregation的思想是博采众长，将可能的hypothesis优势集合起来，将集体智慧融合起来，使预测模型达到更好的效果。

下面先来看一个例子，通过这个例子说明为什么Aggregation能work得更好。

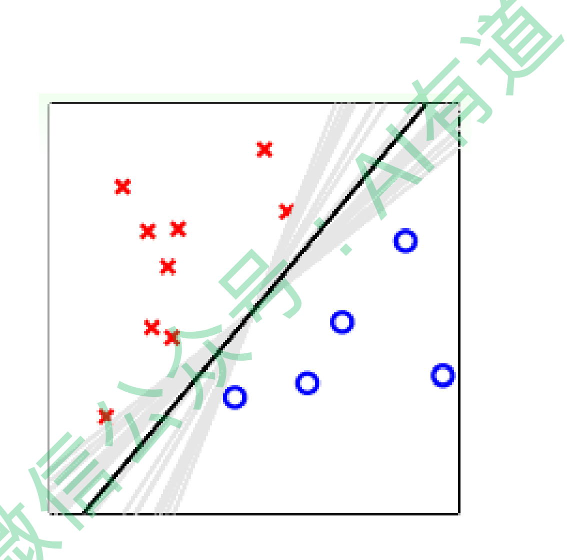


如上图所示，平面上分布着一些待分类的点。如果要求只能用一条水平的线或者垂直的线进行分类，那不论怎么选取直线，都达不到最佳的分类效果。这实际上就是上面介绍的第一种方法：validation。但是，如果可以使用集体智慧，比如一条水平线和两条垂直线组合而成的图中折线形式，就可以将所有的点完全分开，得到了最优化的预测模型。

这个例子表明，通过将不同的hypotheses均匀地结合起来，得到了比单一hypothesis更好的预测模型。这就是aggregation的优势所在，它提高了预测模型的power，起到了特征转换（feature transform）的效果。

- mix **different weak hypotheses** uniformly —$G(\mathbf{x})$ 'strong'
- aggregation $\Longrightarrow$ **feature transform (?)**

我们再从另外一方面来看，同样是平面上分布着一些待分类的点，使用PLA算法，可以得到很多满足条件的分类线，如下图所示：



这无数条PLA选择出来的直线对应的hypothesis都是满足分类要求的。但是我们最想得到的分类直线是中间那条距离所有点都比较远的黑色直线，这与之前SVM目标是一致的。如果我们将所有可能的hypothesis结合起来，以投票的方式进行组合选择，最终会发现投票得到的分类线就是中间和黑色那条。这从哲学的角度来说，就是对各种效果较好的可能性进行组合，得到的结果一般是中庸的、最合适的，即对应图中那条黑色直线。所以，aggregation也起到了正则化（regularization）的效果，让预测模型更具有代表性。



- mix different random-PLA hypotheses uniformly —$G(\mathbf{x})$ 'moderate'
- aggregation $\Longrightarrow$ regularization (?)

基于以上的两个例子，我们得到了aggregation的两个优势：feature transform和regularization。我们之前在机器学习基石课程中就介绍过，feature transform和regularization是对立的，还把它们分别比作踩油门和踩刹车。如果进行feature transform，那么regularization的效果通常很差，反之亦然。也就是说，单一模型通常只能倾向于feature transform和regularization之一，在两者之间做个权衡。但是aggregation却能将feature transform和regularization各自的优势结合起来，好比把油门和刹车都控制得很好，从而得到不错的预测模型。

Uniform Blending

那对于我们已经选择的性能较好的一些矩g_t，如何将它们进行整合、合并，来得到最佳的预测模型呢？这个过程称为blending。

最常用的一种方法是uniform blending，应用于classification分类问题，做法是将每一个可能的矩赋予权重1，进行投票，得到的G(x)表示为：

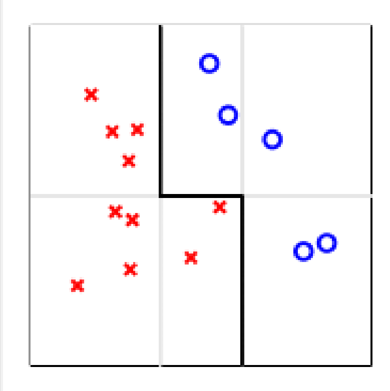
$$g(x)=sign(\sum_{t=1}^{T}1\cdot g_t(x))$$

这种方法对应三种情况：第一种情况是每个候选的矩g_t都完全一样，这跟选其中任意一个g_t效果相同；第二种情况是每个候选的矩g_t都有一些差别，这是最常遇到的，大都可以通过投票的形式使多数意见修正少数意见，从而得到很好的模型，如下图所示；第三种情况是多分类问题，选择投票数最多的那一类即可。



- same g_t (autocracy): as good as one single g_t
- very different g_t (**diversity** + **democracy**): majority can **correct** minority
- similar results with uniform voting for multiclass

$$G(\mathbf{x}) = \underset{1 \le k \le K}{\text{argmax}} \sum_{t=1}^{T} [\![g_t(\mathbf{x}) = k]\!]$$



如果是regression回归问题，uniform blending的做法很简单，就是将所有的矩g_t求平均值：

$$G(x) = \frac{1}{T} \sum_{t=1}^{T} g_t(x)$$

uniform blending for regression对应两种情况：第一种情况是每个候选的矩g_t都完全一样，这跟选其中任意一个g_t效果相同；第二种情况是每个候选的矩g_t都有一些差别，有的$g_t > f(x)$，有的$g_t < f(x)$，此时求平均值的操作可能会消去这种大于和小于的影响，从而得到更好的回归模型。因此，从直觉上来说，求平均值的操作更加稳定，更加准确。

- same g_t (autocracy): as good as one single g_t
- very different g_t (**diversity** + **democracy**):
  some $g_t(\mathbf{x}) > f(\mathbf{x})$, some $g_t(\mathbf{x}) < f(\mathbf{x})$
  $\Longrightarrow$ average **could be** more accurate than individual

对于uniform blending，一般要求每个候选的矩g_t都有一些差别。这样，通过不同矩g_t的组合和集体智慧，都能得到比单一矩g_t更好的模型。

刚才我们提到了uniform blending for regression中，计算g_t的平均值可能比单一的g_t更稳定，更准确。下面进行简单的推导和证明。



$$
\begin{aligned}
\text{avg}\left((g_t(\mathbf{x}) - f(\mathbf{x}))^2\right) &= \text{avg}\left(g_t^2 - 2g_t f + f^2\right) \\
&= \text{avg}\left(g_t^2\right) - 2Gf + f^2 \\
&= \text{avg}\left(g_t^2\right) - G^2 + (G-f)^2 \\
&= \text{avg}\left(g_t^2\right) - 2G^2 + G^2 + (G-f)^2 \\
&= \text{avg}\left(g_t^2 - 2g_t G + G^2\right) + (G-f)^2 \\
&= \text{avg}\left((g_t - G)^2\right) + (G-f)^2
\end{aligned}
$$

推导过程中注意$G(t) = avg(g_t)$。经过推导，我们发现$avg((g_t(x) - f(x))^2)$与$(G-f)^2$之间差了$avg((g_t - G)^2)$项，且是大于零的。从而得到g_t与目标函数f的差值要比G与f的差值大。

刚才是对单一的x进行证明，如果从期望角度，对整个x分布进行上述公式的整理，得到：

$$
\begin{aligned}
\text{avg}\left(E_{\text{out}}(g_t)\right) &= \text{avg}\left(\mathcal{E}(g_t - G)^2\right) + E_{\text{out}}(G) \\
&\geq \qquad\qquad\qquad\quad + E_{\text{out}}(G)
\end{aligned}
$$

从结果上来看，$avg(E_{out}(g_t)) \geq E_{out}(G)$，从而证明了从平均上来说，计算$g_t$的平均值G(t)要比单一的$g_t$更接近目标函数f，regression效果更好。

我们已经知道G是数目为T的g_t的平均值。令包含N个数据的样本D独立同分布于P^N，每次从新的D_t中学习得到新的g_t，在对g_t求平均得到G，当做无限多次，即T趋向于无穷大的时候：

$$
\bar{g} = lim_{T\to\infty}\, G = lim_{T\to\infty}\, \frac{1}{T}\sum_{t=1}^{T} g_t = \epsilon_D A(D)
$$

consider a **virtual** iterative process that for $t = 1, 2, \ldots, T$

1. request size-N data $\mathcal{D}_t$ from P^N (i.i.d.)
2. obtain g_t by $\mathcal{A}(\mathcal{D}_t)$

$$
\bar{g} = \lim_{T\to\infty} G = \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} g_t = \underset{\mathcal{D}}{\mathcal{E}}\ \mathcal{A}(\mathcal{D})
$$



当T趋于无穷大的时候，$G = \bar{g}$，则有如下等式成立：

$$\begin{aligned} \text{avg}\left(E_{\text{out}}(g_t)\right) &= \text{avg}\left(\mathcal{E}(g_t - \bar{g})^2\right) + E_{\text{out}}(\bar{g}) \\ \text{expected performance of } \mathcal{A} &= \text{expected deviation to consensus} \\ &\quad + \text{performance of consensus} \end{aligned}$$

- performance of consensus: called **bias**
- expected deviation to consensus: called **variance**

上述等式中左边表示演算法误差的期望值；右边第二项表示不同g_t的平均误差共识，用偏差bias表示；右边第一项表示不同g_t与共识的差距是多少，反映g_t之间的偏差，用方差variance表示。也就是说，一个演算法的平均表现可以被拆成两项，一个是所有g_t的共识，一个是不同g_t之间的差距是多少，即bias和variance。而uniform blending的操作时求平均的过程，这样就削减弱化了上式第一项variance的值，从而演算法的表现就更好了，能得到更加稳定的表现。

Linear and Any Blending

上一部分讲的是uniform blending，即每个g_t所占的权重都是1，求平均的思想。下面我们将介绍linear blending，每个g_t赋予的权重α_t并不相同，其中$\alpha_t \geq 0$。我们最终得到的预测结果等于所有g_t的线性组合。

linear blending: known g_t, each to be given α_t ballot

$$G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \cdot g_t(\mathbf{x})\right) \text{ with } \alpha_t \geq 0$$

如何确定α_t的值，方法是利用误差最小化的思想，找出最佳的α_t，使$E_{in}(\alpha)$取最小值。例如对于linear blending for regression，$E_{in}(\alpha)$可以写成下图左边形式，其中α_t是带求解参数，$g_t(x_n)$是每个矩得到的预测值，由已知矩得到。这种形式很类似于下图右边的形式，即加上特征转换$\phi_i(x_n)$的linear regression模型。两个式子中的$g_t(x_n)$对应于$\phi_i(x_n)$，唯一不同的就是linear blending for regression中$\alpha_t \geq 0$，而linear regression中w_i没有限制。



computing 'good' α_t : $\min_{\alpha_t \geq 0} E_{in}(\alpha)$

linear blending for regression

$$\min_{\alpha_t \geq 0} \frac{1}{N} \sum_{n=1}^{N} \left(y_n - \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n) \right)^2$$

LinReg + transformation

$$\min_{w_i} \frac{1}{N} \sum_{n=1}^{N} \left(y_n - \sum_{i=1}^{\tilde{d}} w_i \phi_i(\mathbf{x}_n) \right)^2$$

like two-level learning, remember? :-)

linear blending = LinModel + hypotheses as transform + constraints

这种求解α_t的方法就像是使用two-level learning，类似于我们之前介绍的probabilistic SVM。这里，我们先计算$g_t(x_n)$，再进行linear regression得到α_t值。总的来说，linear blending由三个部分组成：LinModel，hypotheses as transform，constraints。其中值得注意的一点就是，计算过程中可以把g_t当成feature transform，求解过程就跟之前没有什么不同，除了$\alpha \geq 0$的条件限制。

linear blending = LinModel + hypotheses as transform + constraints:

$$\min_{\alpha_t \geq 0} \frac{1}{N} \sum_{n=1}^{N} \mathrm{err}\left(y_n, \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n) \right)$$

我们来看一下linear blending中的constraint $\alpha_t \geq 0$。这个条件是否一定要成立呢？如果$\alpha_t < 0$，会带来什么后果呢？其实$\alpha_t < 0$并不会影响分类效果，只需要将正类看成负类，负类当成正类即可。例如分类问题，判断该点是正类对应的$\alpha_t < 0$，则它就表示该点是负类，且对应的$-\alpha_t > 0$。如果我们说这个样本是正类的概率是-99%，意思也就是说该样本是负类的概率是99%。$\alpha_t \geq 0$和$\alpha_t < 0$的效果是等同的一致的。所以，我们可以把$\alpha_t \geq 0$这个条件舍去，这样linear blending就可以使用常规方法求解。



linear blending for binary classification

$$\text{if } \alpha_t < 0 \implies \alpha_t g_t(\mathbf{x}) = |\alpha_t| \left(-g_t(\mathbf{x})\right)$$

- negative α_t for g_t ≡ positive $|\alpha_t|$ for $-g_t$
- **if you have a stock up/down classifier with 99% error, tell me! :-)**

in practice, often
linear blending = LinModel + hypotheses as transform ~~+ constraints~~

Linear Blending中使用的g_t是通过模型选择而得到的，利用validation，从D_{train}中得到$g_1^-, g_2^-, \cdots, g_T^-$。然后将$D_{train}$中每个数据点经过各个矩的计算得到的值，代入到相应的linear blending计算公式中，迭代优化得到对应α值。最终，再利用所有样本数据，得到新的g_t代替g_t^-，则G(t)就是g_t的线性组合而不是g_t^-，系数是α_t。

in practice, often

$$g_1 \in \mathcal{H}_1, g_2 \in \mathcal{H}_2, \ldots, g_T \in \mathcal{H}_T$$

by minimum E_{in}

- recall: **selection by minimum** E_{in}
  —best of best, paying $d_{VC}\left(\bigcup_{t=1}^{T} \mathcal{H}_t\right)$
- recall: linear blending includes selection as special case
  —by setting $\alpha_t = [\![E_{val}(g_t^-) \text{ smallest}]\!]$
- complexity price of linear blending with E_{in} (aggregation of best):
  $\geq d_{VC}\left(\bigcup_{t=1}^{T} \mathcal{H}_t\right)$

like selection, blending practically done with
(E_{val} instead of E_{in}) + (g_t^- from minimum E_{train})



Given $g_1^-, g_2^-, \ldots, g_T^-$ from $\mathcal{D}_{\text{train}}$, transform $(\mathbf{x}_n, y_n)$ in $\mathcal{D}_{\text{val}}$ to $(\mathbf{z}_n = \mathbf{\Phi}^-(\mathbf{x}_n), y_n)$, where $\mathbf{\Phi}^-(\mathbf{x}) = (g_1^-(\mathbf{x}), \ldots, g_T^-(\mathbf{x}))$

Linear Blending	Any Blending (Stacking)
1 compute $\alpha = \text{LinearModel}\left(\{(\mathbf{z}_n, y_n)\}\right)$	1 compute $\tilde{g} = \text{AnyModel}\left(\{(\mathbf{z}_n, y_n)\}\right)$
2 return $G_{\text{LINB}}(\mathbf{x}) = \text{LinearHypothesis}_\alpha(\mathbf{\Phi}(\mathbf{x}))$,	2 return $G_{\text{ANYB}}(\mathbf{x}) = \tilde{g}(\mathbf{\Phi}(\mathbf{x}))$,

where $\mathbf{\Phi}(\mathbf{x}) = (g_1(\mathbf{x}), \ldots, g_T(\mathbf{x}))$

any blending:

- powerful, achieves conditional blending
- but danger of overfitting, as always :-(

除了linear blending之外，还可以使用任意形式的blending。linear blending中，G(t)是g(t)的线性组合；any blending中，G(t)可以是g(t)的任何函数形式（非线性）。这种形式的blending也叫做Stacking。any blending的优点是模型复杂度提高，更容易获得更好的预测模型；缺点是复杂模型也容易带来过拟合的危险。所以，在使用any blending的过程中要时刻注意避免过拟合发生，通过采用regularization的方法，让模型具有更好的泛化能力。

Bagging(Bootstrap Aggregation)

总结一些上面讲的内容，blending的做法就是将已经得到的矩g_t进行aggregate的操作。具体的aggregation形式包括：uniform，non-uniforn和conditional。

blending: aggregate after getting g_t;
learning: aggregate as well as getting g_t

aggregation type	blending	learning
uniform	voting/averaging	?
non-uniform	linear	?
conditional	stacking	?

现在考虑一个问题：如何得到不同的g_t呢？可以选取不同模型H；可以设置不同的参数，例如η、迭代次数n等；可以由算法的随机性得到，例如PLA、随机种子等；可以选择不同的数据样本等。这些方法都可能得到不同的g_t。

learning g_t for uniform aggregation: diversity important

- diversity by different models: $g_1 \in \mathcal{H}_1, g_2 \in \mathcal{H}_2, \ldots, g_T \in \mathcal{H}_T$
- diversity by different parameters: GD with $\eta = 0.001, 0.01, \ldots, 10$
- diversity by algorithmic randomness:
  random PLA with different random seeds
- diversity by data randomness:
  within-cross-validation hypotheses g_v^-

那如何利用已有的一份数据集来构造出不同的g_t呢？首先，我们回顾一下之前介绍的bias-variance，即一个演算法的平均表现可以被拆成两项，一个是所有g_t的共识（bias），一个是不同g_t之间的差距是多少（variance）。其中每个g_t都是需要新的数据集的。只有一份数据集的情况下，如何构造新的数据集？

$$\begin{aligned} \text{expected performance of } \mathcal{A} &= \text{expected deviation to consensus} \\ &\quad + \text{performance of consensus} \\ \text{consensus } \bar{g} &= \text{expected } g_t \text{ from } \mathcal{D}_t \sim P^N \end{aligned}$$

其中，$\bar{g}$是在矩个数T趋向于无穷大的时候，不同的g_t计算平均得到的值。这里我们为了得到$\bar{g}$，做两个近似条件：

- **有限的T；**
- **由已有数据集D构造出D_t P^N，独立同分布**

第一个条件没有问题，第二个近似条件的做法就是bootstrapping。bootstrapping是统计学的一个工具，思想就是从已有数据集D中模拟出其他类似的样本D_t。

- consensus more stable than direct $\mathcal{A}(\mathcal{D})$,
  but comes from many more $\mathcal{D}_t$ than the $\mathcal{D}$ on hand
- want: approximate $\bar{g}$ by
  - finite (large) T
  - approximate $g_t = \mathcal{A}(\mathcal{D}_t)$ from $\mathcal{D}_t \sim P^N$ using only $\mathcal{D}$

bootstrapping: a statistical tool that
re-samples from $\mathcal{D}$ to 'simulate' $\mathcal{D}_t$



bootstrapping的做法是，假设有N笔资料，先从中选出一个样本，再放回去，再选择一个样本，再放回去，共重复N次。这样我们就得到了一个新的N笔资料，这个新的$\breve{D}_t$中可能包含原D里的重复样本点，也可能没有原D里的某些样本，$\breve{D}_t$与D类似但又不完全相同。值得一提的是，抽取-放回的操作不一定非要是N，次数可以任意设定。例如原始样本有10000个，我们可以抽取-放回3000次，得到包含3000个样本的$\breve{D}_t$也是完全可以的。利用bootstrap进行aggragation的操作就被称为bagging。

virtual aggregation

consider a **virtual** iterative process that for $t = 1, 2, \ldots, T$

1. request size-N data $\mathcal{D}_t$ from P^N (i.i.d.)
2. obtain g_t by $\mathcal{A}(\mathcal{D}_t)$

$G = \text{Uniform}(\{g_t\})$

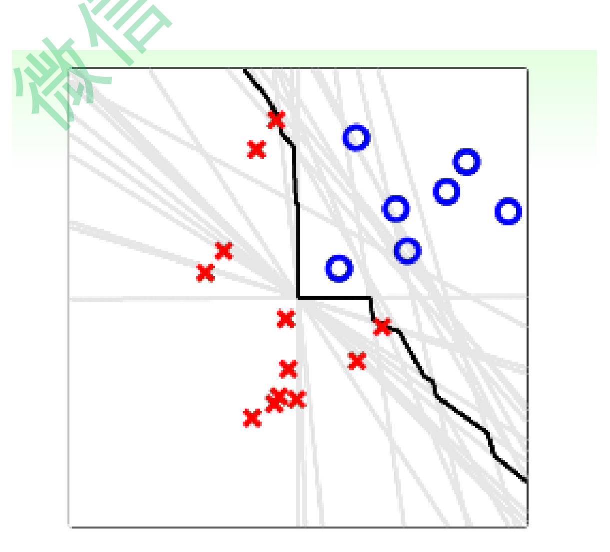
bootstrap aggregation

consider a **physical** iterative process that for $t = 1, 2, \ldots, T$

1. request size-N' data $\tilde{\mathcal{D}}_t$ from bootstrapping
2. obtain g_t by $\mathcal{A}(\tilde{\mathcal{D}}_t)$

$G = \text{Uniform}(\{g_t\})$

下面举个实际中Bagging Pocket算法的例子。如下图所示，先通过bootstrapping得到25个不同样本集，再使用pocket算法得到25个不同的g_t，每个pocket算法迭代1000次。最后，再利用blending，将所有的g_t融合起来，得到最终的分类线，如图中黑线所示。可以看出，虽然bootstrapping会得到差别很大的分类线（灰线），但是经过blending后，得到的分类线效果是不错的，则bagging通常能得到最佳的分类模型。



$T_{\text{POCKET}} = 1000;\ T_{\text{BAG}} = 25$

值得注意的是，只有当演算法对数据样本分布比较敏感的情况下，才有比较好的表现。



总结

本节课主要介绍了blending和bagging的方法，它们都属于aggregation，即将不同的g_t合并起来，利用集体的智慧得到更加优化的G(t)。Blending通常分为三种情况：Uniform Blending，Linear Blending和Any Blending。其中，uniform blending采样最简单的“一人一票”的方法，linear blending和any blending都采用标准的two-level learning方法，类似于特征转换的操作，来得到不同g_t的线性组合或非线性组合。最后，我们介绍了如何利用bagging（bootstrap aggregation），从已有数据集D中模拟出其他类似的样本D_t，而得到不同的g_t，再合并起来，优化预测模型。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程





林轩田《机器学习技法》课程笔记8 -- Adaptive Boosting

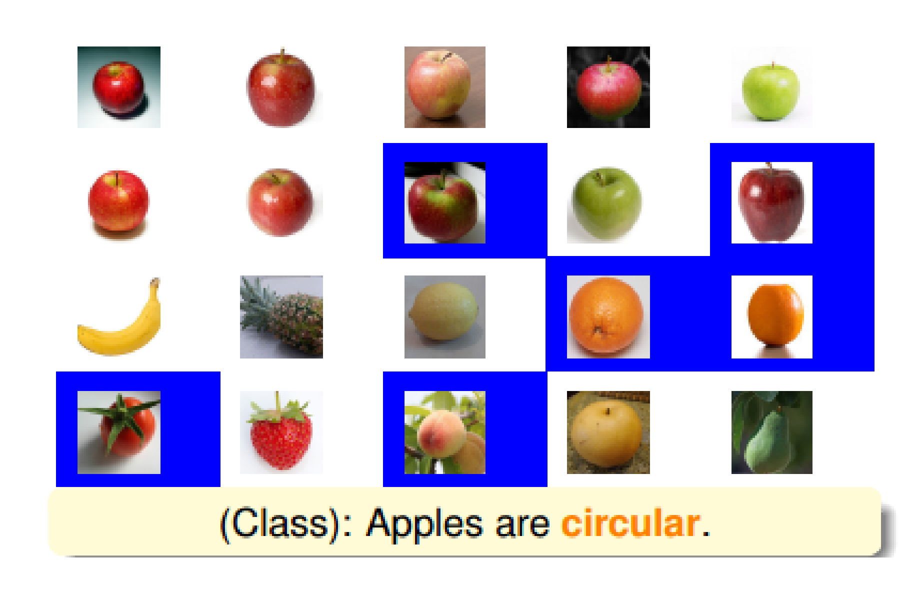
作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要开始介绍Aggregation Models，目的是将不同的hypothesis得到的g_t集合起来，利用集体智慧得到更好的预测模型G。首先我们介绍了Blending，blending是将已存在的所有g_t结合起来，可以是uniformly，linearly，或者non-linearly组合形式。然后，我们讨论了在没有那么多g_t的情况下，使用bootstrap方式，从已有数据集中得到新的类似的数据集，从而得到不同的g_t。这种做法称为bagging。本节课将继续从这些概念出发，介绍一种新的演算法。

Motivation of Boosting

我们先来看一个简单的识别苹果的例子，老师展示20张图片，让6岁孩子们通过观察，判断其中哪些图片的内容是苹果。从判断的过程中推导如何解决二元分类问题的方法。

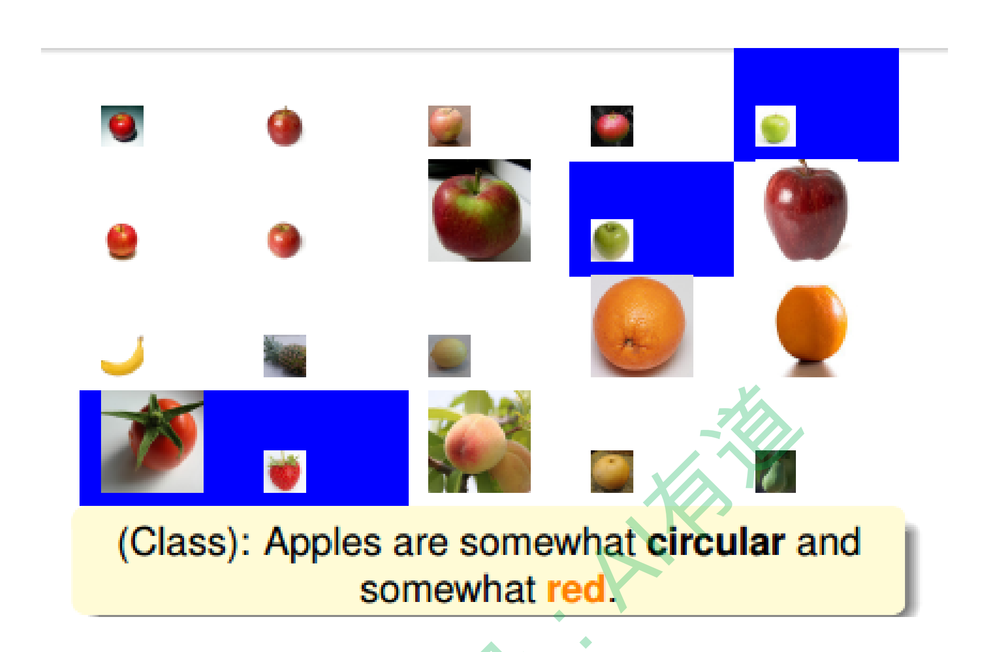
显然这是一个监督式学习，20张图片包括它的标签都是已知的。首先，学生Michael回答说：所有的苹果应该是圆形的。根据Michael的判断，对应到20张图片中去，大部分苹果能被识别出来，但也有错误。其中错误包括有的苹果不是圆形，而且圆形的水果也不一定是苹果。如下图所示：





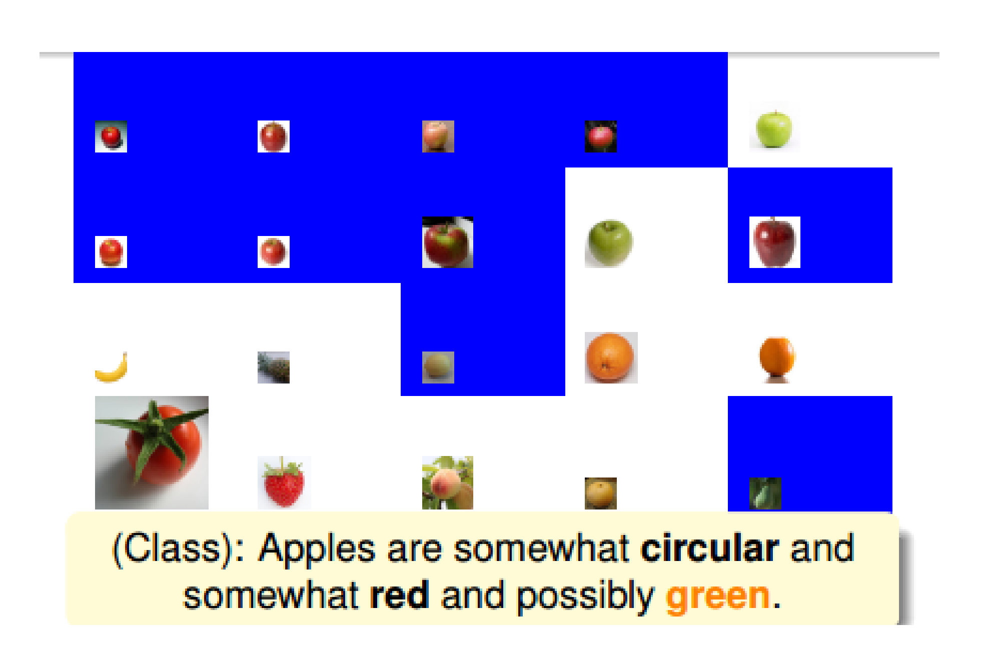
上图中蓝色区域的图片代表分类错误。显然，只用“苹果是圆形的”这一个条件不能保证分类效果很好。我们把蓝色区域（分类错误的图片）放大，分类正确的图片缩小，这样在接下来的分类中就会更加注重这些错误样本。

然后，学生Tina观察被放大的错误样本和上一轮被缩小的正确样本，回答说：苹果应该是红色的。根据Tina的判断，得到的结果如下图所示：



上图中蓝色区域的图片一样代表分类错误，即根据这个苹果是红色的条件，使得青苹果和草莓、西红柿都出现了判断错误。那么结果就是把这些分类错误的样本放大化，其它正确的样本缩小化。同样，这样在接下来的分类中就会更加注重这些错误样本。

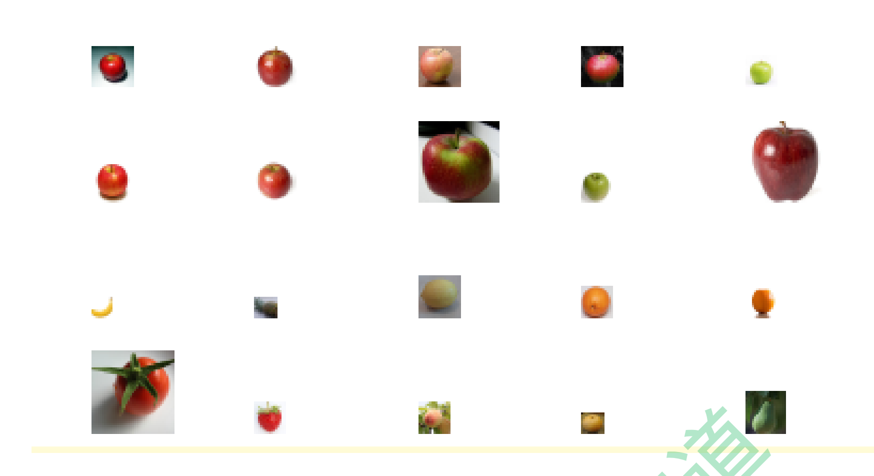
接着，学生Joey经过观察又说：苹果也可能是绿色的。根据Joey的判断，得到的结果如下图所示：



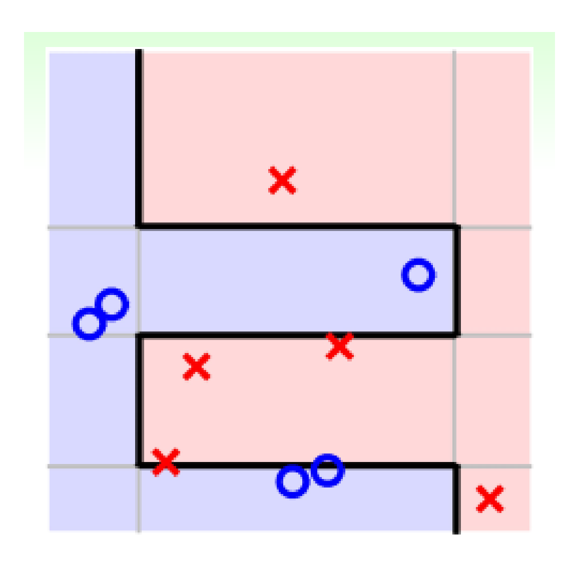


上图中蓝色区域的图片一样代表分类错误，根据苹果是绿色的条件，使得图中蓝色区域都出现了判断错误。同样把这些分类错误的样本放大化，其它正确的样本缩小化，在下一轮判断继续对其修正。

后来，学生Jessica又发现：上面有梗的才是苹果。得到如下结果：



经过这几个同学的推论，苹果被定义为：圆的，红色的，也可能是绿色的，上面有梗。从一个一个的推导过程中，我们似乎得到一个较为准确的苹果的定义。虽然可能不是非常准确，但是要比单一的条件要好得多。也就是说把所有学生对苹果的定义融合起来，最终得到一个比较好的对苹果的总体定义。这种做法就是我们本节课将要讨论的演算法。这些学生代表的就是简单的hypotheses g_t，将所有g_t融合，得到很好的预测模型G。例如，二维平面上简单的hypotheses（水平线和垂直线），这些简单g_t最终组成的较复杂的分类线能够较好地将正负样本完全分开，即得到了好的预测模型。



所以，上个苹果的例子中，不同的学生代表不同的hypotheses g_t；最终得到的苹果总体定义就代表hypothesis G；而老师就代表演算法A，指导学生的注意力集中到关键的

例子中（错误样本），从而得到更好的苹果定义。其中的数学原理，我们下一部分详细介绍。

- students: simple hypotheses g_t (like vertical/horizontal lines)
- (Class): sophisticated hypothesis G (like black curve)
- Teacher: a tactic learning algorithm that **directs the students to focus on key examples**

Diversity by Re-weighting

在介绍这个演算法之前，我们先来讲一下上节课就介绍过的bagging。Bagging的核心是bootstrapping，通过对原始数据集D不断进行bootstrap的抽样动作，得到与D类似的数据集$\hat{D}_t$，每组$\hat{D}_t$都能得到相应的g_t，从而进行aggregation的操作。现在，假如包含四个样本的D经过bootstrap，得到新的$\hat{D}_t$如下：

$$\mathcal{D} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_3, y_3), (\mathbf{x}_4, y_4)\}$$

$$\xrightarrow{\text{bootstrap}} \tilde{\mathcal{D}}_t = \{(\mathbf{x}_1, y_1), (\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), (\mathbf{x}_4, y_4)\}$$

那么，对于新的$\hat{D}_t$，把它交给base algorithm，找出E_{in}最小时对应的g_t，如下图右边所示。

$$E_{in}^{0/1}(h) = \frac{1}{4}\sum_{n=1}^{4}[y \neq h(x)]$$

由于$\hat{D}_t$完全是D经过bootstrap得到的，其中样本(x_1, y_1)出现2次，(x_2, y_2)出现1次，(x_3, y_3)出现0次，(x_4, y_4)出现1次。引入一个参数u_i来表示原D中第i个样本在$\hat{D}_t$中出现的次数，如下图左边所示。

$$E_{in}^{u}(h) = \frac{1}{4}\sum_{n=1}^{4}u_n^{(t)} \cdot [y_n \neq h(x)]$$



weighted E_{in} on $\mathcal{D}$

$$E_{\text{in}}^{\mathbf{u}}(h) = \frac{1}{4}\sum_{n=1}^{4} u_n^{(t)} \cdot [\![y_n \neq h(\mathbf{x}_n)]\!]$$

$(\mathbf{x}_1, y_1)$, $u_1 = 2$
$(\mathbf{x}_2, y_2)$, $u_2 = 1$
$(\mathbf{x}_3, y_3)$, $u_3 = 0$
$(\mathbf{x}_4, y_4)$, $u_4 = 1$

E_{in} on $\tilde{\mathcal{D}}_t$

$$E_{\text{in}}^{0/1}(h) = \frac{1}{4}\sum_{(\mathbf{x},y)\in\tilde{\mathcal{D}}_t} [\![y \neq h(\mathbf{x})]\!]$$

$(\mathbf{x}_1, y_1)$, $(\mathbf{x}_1, y_1)$
$(\mathbf{x}_2, y_2)$

$(\mathbf{x}_4, y_4)$

参数u相当于是权重因子，当$\hat{D}_t$中第i个样本出现的次数越多的时候，那么对应的u_i越大，表示在error function中对该样本的惩罚越多。所以，从另外一个角度来看bagging，它其实就是通过bootstrap的方式，来得到这些u_i值，作为犯错样本的权重因子，再用base algorithn最小化包含u_i的error function，得到不同的g_t。这个error function被称为bootstrap-weighted error。

这种算法叫做Weightd Base Algorithm，目的就是最小化bootstrap-weighted error。

minimize (regularized)

$$E_{\text{in}}^{\mathbf{u}}(h) = \frac{1}{N}\sum_{n=1}^{N} u_n \cdot \text{err}(y_n, h(\mathbf{x}_n))$$

其实，这种weightd base algorithm我们之前就介绍过类似的算法形式。例如在soft-margin SVM中，我们引入允许犯错的项，同样可以将每个点的error乘以权重因子u_n。加上该项前的参数C，经过QP，最终得到$0 \leq \alpha_n \leq Cu_n$，有别于之前介绍的$0 \leq \alpha_n \leq C$。这里的$u_n$相当于每个犯错的样本的惩罚因子，并会反映到$\alpha_n$的范围限定上。

同样在logistic regression中，同样可以对每个犯错误的样本乘以相应的u_n，作为惩罚因子。u_n表示该错误点出现的次数，u_n越大，则对应的惩罚因子越大，则在最小化error时就应该更加重视这些点。



SVM

$E_{in}^{\mathbf{u}} \propto C \sum_{n=1}^{N} u_n \widehat{\text{err}}_{\text{SVM}}$ by dual QP

$\Leftrightarrow$ adjusted upper bound

$0 \leq \alpha_n \leq C u_n$

logistic regression

$E_{in}^{\mathbf{u}} \propto \sum_{n=1}^{N} u_n \text{err}_{\text{CE}}$ by SGD

$\Leftrightarrow$ sample $(\mathbf{x}_n, y_n)$ with probability proportional to u_n

其实这种example-weighted learning，我们在机器学习基石课程第8次笔记中就介绍过class-weighted的思想。二者道理是相通的。

知道了u的概念后，我们知道不同的u组合经过base algorithm得到不同的g_t。那么如何选取u，使得到的g_t之间有很大的不同呢？之所以要让所有的g_t差别很大，是因为上节课aggregation中，我们介绍过g_t越不一样，其aggregation的效果越好，即每个人的意见越不相同，越能运用集体的智慧，得到好的预测模型。

为了得到不同的g_t，我们先来看看g_t和g_{t+1}是怎么得到的：

$$g_t \leftarrow \underset{h \in \mathcal{H}}{\text{argmin}} \left(\sum_{n=1}^{N} u_n^{(t)} [\![y_n \neq h(\mathbf{x}_n)]\!] \right)$$

$$g_{t+1} \leftarrow \underset{h \in \mathcal{H}}{\text{argmin}} \left(\sum_{n=1}^{N} u_n^{(t+1)} [\![y_n \neq h(\mathbf{x}_n)]\!] \right)$$

if g_t 'not good' for $\mathbf{u}^{(t+1)}$ $\Longrightarrow$ g_t-like hypotheses not returned as g_{t+1}
$\Longrightarrow$ g_{t+1} diverse from g_t

如上所示，g_t是由u_n^t得到的，g_{t+1}是由$u_n^{(t+1)}$得到的。如果g_t这个模型在使用$u_n^{(t+1)}$的时候得到的error很大，即预测效果非常不好，那就表示由$u_n^{(t+1)}$计算的g_{t+1}会与g_t有很大不同。而g_{t+1}与g_t差异性大正是我们希望看到的。

怎么做呢？方法是利用g_t在使用$u_n^{(t+1)}$的时候表现很差的条件，越差越好。如果在g_t作用下，$u_n^{(t+1)}$中的表现（即error）近似为0.5的时候，表明g_t对$u_n^{(t+1)}$的预测分类没有什么作用，就像抛硬币一样，是随机选择的。这样的做法就能最大限度地保证g_{t+1}会与g_t有较大的差异性。其数学表达式如下所示：



idea: construct $\mathbf{u}^{(t+1)}$ to make g_t random-like

$$\frac{\sum_{n=1}^{N} u_n^{(t+1)} [\![y_n \neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^{N} u_n^{(t+1)}} = \frac{1}{2}$$

乍看上面这个式子，似乎不好求解。但是，我们对它做一些等价处理，其中分式中分子可以看成g_t作用下犯错误的点，而分母可以看成犯错的点和没有犯错误的点的集合，即所有样本点。其中犯错误的点和没有犯错误的点分别用橘色方块和绿色圆圈表示：

$$\text{want: } \frac{\sum_{n=1}^{N} u_n^{(t+1)} [\![y_n \neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^{N} u_n^{(t+1)}} = \frac{\blacksquare_{t+1}}{\blacksquare_{t+1} + \bullet_{t+1}} = \frac{1}{2}, \text{ where}$$

$$\blacksquare_{t+1} = \sum_{n=1}^{N} u_n^{(t+1)} [\![y_n \neq g_t(\mathbf{x}_n)]\!], \bullet_{t+1} = \sum_{n=1}^{N} u_n^{(t+1)} [\![y_n = g_t(\mathbf{x}_n)]\!]$$

要让分式等于0.5，显然只要将犯错误的点和没有犯错误的点的数量调成一样就可以了。也就是说，在g_t作用下，让犯错的$u_n^{(t+1)}$数量和没有犯错的$u_n^{(t+1)}$数量一致就行（包含权重u_n^{t+1}）。一种简单的方法就是利用放大和缩小的思想（本节课开始引入识别苹果的例子中提到的放大图片和缩小图片就是这个目的），将犯错误的u_n^t和没有犯错误的u_n^t做相应的乘积操作，使得二者值变成相等。例如u_n^t of incorrect为1126，u_n^t of correct为6211，要让$u_n^{(t+1)}$中错误比例正好是0.5，可以这样做，对于incorrect $u_n^{(t+1)}$：

$$u_n^{(t+1)} \leftarrow u_n^{(t)} \cdot 6211$$

对于correct $u_n^{(t+1)}$：

$$u_n^{(t+1)} \leftarrow u_n^{(t)} \cdot 1126$$

或者利用犯错的比例来做，令weighted incorrect rate和weighted correct rate分别设为$\frac{1126}{7337}$和$\frac{6211}{7337}$。一般求解方式是令犯错率为ϵ_t，在计算$u_n^{(t+1)}$的时候，u_n^t分别乘以$(1-\epsilon_t)$和ϵ_t。



- need: (total $u_n^{(t+1)}$ of incorrect) = (total $u_n^{(t+1)}$ of correct)
  $\blacksquare_{t+1}$ $\bullet_{t+1}$
- one possibility by **re-scaling (multiplying) weights**, if

(total $u_n^{(t)}$ of incorrect) = 1126 ; (weighted incorrect rate) = $\frac{1126}{7337}$	(total $u_n^{(t)}$ of correct) = 6211 ; (weighted correct rate) = $\frac{6211}{7337}$
incorrect: $u_n^{(t+1)} \leftarrow u_n^{(t)} \cdot 6211$	correct: $u_n^{(t+1)} \leftarrow u_n^{(t)} \cdot 1126$

'optimal' re-weighting under weighted incorrect rate ϵ_t:
multiply incorrect $\propto (1 - \epsilon_t)$; multiply correct $\propto \epsilon_t$

Adaptive Boosting Algorithm

上一部分，我们介绍了在计算$u_n^{(t+1)}$的时候，u_n^t分别乘以$(1-\epsilon_t)$和ϵ_t。下面将构造一个新的尺度因子：

$$\diamond t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$$

那么引入这个新的尺度因子之后，对于错误的u_n^t，将它乘以$\diamond t$；对于正确的u_n^t，将它除以$\diamond t$。这种操作跟之前介绍的分别乘以$(1-\epsilon_t)$和ϵ_t的效果是一样的。之所以引入$\diamond t$是因为它告诉我们更多的物理意义。因为如果$\epsilon_t \leq \frac{1}{2}$，得到$\diamond t \geq 1$，那么接下来错误的$u_n^t$与$\diamond t$的乘积就相当于把错误点放大了，而正确的$u_n^t$与$\diamond t$的相除就相当于把正确点缩小了。这种scale up incorrect和scale down correct的做法与本节课开始介绍的学生识别苹果的例子中放大错误的图片和缩小正确的图片是一个原理，让学生能够将注意力更多地放在犯错误的点上。通过这种scaling-up incorrect的操作，能够保证得到不同于g_t的g_{t+1}。

define scaling factor $\blacklozenge_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$

$$\text{incorrect} \leftarrow \text{incorrect} \cdot \blacklozenge_t$$
$$\text{correct} \leftarrow \text{correct} \;/\; \blacklozenge_t$$

- **equivalent** to optimal re-weighting
- $\blacklozenge_t \geq 1$ iff $\epsilon_t \leq \frac{1}{2}$
  —physical meaning: **scale up incorrect**; **scale down correct**
  —like what Teacher does



值得注意的是上述的结论是建立在$\epsilon_t \leq \frac{1}{2}$的基础上，如果$\epsilon_t \geq \frac{1}{2}$，那么就做相反的推论即可。关于$\epsilon_t \geq \frac{1}{2}$的情况，我们稍后会进行说明。

从这个概念出发，我们可以得到一个初步的演算法。其核心步骤是每次迭代时，利用$\diamond t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$把$u_t$更新为$u_{t+1}$。具体迭代步骤如下：

$\mathbf{u}^{(1)} = ?$
for $t = 1, 2, \ldots, T$

1. obtain g_t by $\mathcal{A}(\mathcal{D}, \mathbf{u}^{(t)})$, where $\mathcal{A}$ tries to minimize $\mathbf{u}^{(t)}$-weighted 0/1 error
2. update $\mathbf{u}^{(t)}$ to $\mathbf{u}^{(t+1)}$ by $\blacklozenge_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$, where ϵ_t = weighted error (incorrect) rate of g_t

return $G(\mathbf{x}) = ?$

但是，上述步骤还有两个问题没有解决，第一个问题是初始的$u^{(1)}$应为多少呢？一般来说，为了保证第一次E_{in}最小的话，设$u^{(1)} = \frac{1}{N}$即可。这样最开始的g_1就能由此推导。第二个问题，最终的G(x)应该怎么求？是将所有的g(t)合并uniform在一起吗？一般来说并不是这样直接uniform求解，因为g_{t+1}是通过g_t得来的，二者在E_{in}上的表现差别比较大。所以，一般是对所有的g(t)进行linear或者non-linear组合来得到G(t)。

- want g_1 'best' for E_{in}: $u_n^{(1)} = \frac{1}{N}$
- $G(\mathbf{x})$:
  - uniform? but g_2 very bad for E_{in} (why? :-))
  - linear, non-linear? as you wish

接下来的内容，我们将对上面的第二个问题进行探讨，研究一种算法，将所有的g(t)进行linear组合。方法是计算$g(t)$的同时，就能计算得到其线性组合系数α_t，即aggregate linearly on the fly。这种算法使最终求得g_{t+1}的时候，所有g_t的线性组合系数α也求得了，不用再重新计算α了。这种Linear Aggregation on the Fly算法流程为：



$\mathbf{u}^{(1)} = [\frac{1}{N}, \frac{1}{N}, \cdots, \frac{1}{N}]$
for $t = 1, 2, \ldots, T$

1. obtain g_t by $\mathcal{A}(\mathcal{D}, \mathbf{u}^{(t)})$, where ...
2. update $\mathbf{u}^{(t)}$ to $\mathbf{u}^{(t+1)}$ by $\blacklozenge_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$, where ...
3. compute $\alpha_t = \ln(\blacklozenge_t)$

return $G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})\right)$

如何在每次迭代的时候计算α_t呢？我们知道α_t与ϵ_t是相关的：ϵ_t越小，对应的α_t应该越大，ϵ_t越大，对应的α_t应该越小。又因为$\diamond t$与ϵ_t是正相关的，所以，α_t应该是$\diamond t$的单调函数。我们构造α_t为：

$$\alpha_t = ln(\diamond t)$$

α_t这样取值是有物理意义的，例如当$\epsilon_t = \frac{1}{2}$时，error很大，跟掷骰子这样的随机过程没什么两样，此时对应的$\diamond t = 1$，$\alpha_t = 0$，即此g_t对G没有什么贡献，权重应该设为零。而当$\epsilon_t = 0$时，没有error，表示该g_t预测非常准，此时对应的$\diamond t = \infty$，$\alpha_t = \infty$，即此g_t对G贡献非常大，权重应该设为无穷大。

- wish: large α_t for 'good' $g_t \Longleftarrow \alpha_t = \text{monotonic}(\blacklozenge_t)$
- will take $\alpha_t = \ln(\blacklozenge_t)$
  - $\epsilon_t = \frac{1}{2} \Longrightarrow \blacklozenge_t = 1 \Longrightarrow \alpha_t = 0$ (bad g_t zero weight)
  - $\epsilon_t = 0 \Longrightarrow \blacklozenge_t = \infty \Longrightarrow \alpha_t = \infty$ (super g_t superior weight)

这种算法被称为Adaptive Boosting。它由三部分构成：base learning algorithm A，re-weighting factor $\diamond t$和linear aggregation α_t。这三部分分别对应于我们在本节课开始介绍的例子中的Student，Teacher和Class。

Adaptive Boosting = weak base learning algorithm $\mathcal{A}$ (Student)
+ optimal re-weighting factor $\blacklozenge_t$ (Teacher)
+ 'magic' linear aggregation α_t (Class)

综上所述，完整的adaptive boosting（AdaBoost）Algorithm流程如下：



$\mathbf{u}^{(1)} = [\frac{1}{N}, \frac{1}{N}, \cdots, \frac{1}{N}]$

for $t = 1, 2, \ldots, T$

1. obtain g_t by $\mathcal{A}(\mathcal{D}, \mathbf{u}^{(t)})$, where $\mathcal{A}$ tries to minimize $\mathbf{u}^{(t)}$-weighted 0/1 error
2. update $\mathbf{u}^{(t)}$ to $\mathbf{u}^{(t+1)}$ by

   $[\![y_n \neq g_t(\mathbf{x}_n)]\!]$ (incorrect examples): $u_n^{(t+1)} \leftarrow u_n^{(t)} \cdot \blacklozenge_t$

   $[\![y_n = g_t(\mathbf{x}_n)]\!]$ (correct examples): $u_n^{(t+1)} \leftarrow u_n^{(t)} / \blacklozenge_t$

   where $\blacklozenge_t = \sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$ and $\epsilon_t = \frac{\sum_{n=1}^{N} u_n^{(t)} [\![y_n \neq g_t(\mathbf{x}_n)]\!]}{\sum_{n=1}^{N} u_n^{(t)}}$
3. compute $\alpha_t = \ln(\blacklozenge_t)$

return $G(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})\right)$

从我们之前介绍过的VC bound角度来看，AdaBoost算法理论上满足：

- From VC bound

$$E_{\text{out}}(G) \leq E_{\text{in}}(G) + O\left(\sqrt{\underbrace{O(d_{\text{VC}}(\mathcal{H}) \cdot T \log T)}_{d_{\text{VC}} \text{ of all possible } G} \cdot \frac{\log N}{N}}\right)$$

- **first term can be small**: $E_{\text{in}}(G) = 0$ after $T = O(\log N)$ iterations if $\epsilon_t \leq \epsilon < \frac{1}{2}$ always
- **second term can be small**: overall d_{VC} grows "slowly" with T

上式中，$E_{out}(G)$的上界由两部分组成，一项是$E_{in}(G)$，另一项是模型复杂度O(*)。模型复杂度中$d_{vc}(H)$是g_t的VC Dimension，T是迭代次数，可以证明G的d_{vc}服从$O(d_{vc}(H) \cdot T log\, T)$。

对这个VC bound中的第一项$E_{in}(G)$来说，有一个很好的性质：如果满足$\epsilon_t \leq \epsilon < \frac{1}{2}$，则经过$T = O(log\, N)$次迭代之后，$E_{in}(G)$能减小到等于零的程度。而当N很大的时候，其中第二项也能变得很小。因为这两项都能变得很小，那么整个$E_{out}(G)$就能被限定在一个有限的上界中。

其实，这种性质也正是AdaBoost算法的精髓所在。只要每次的$\epsilon_t \leq \epsilon < \frac{1}{2}$，即所选择的矩g比乱猜的表现好一点点，那么经过每次迭代之后，矩g的表现都会比原来更好一些，逐渐变强，最终得到$E_{in} = 0$且E_{out}很小。

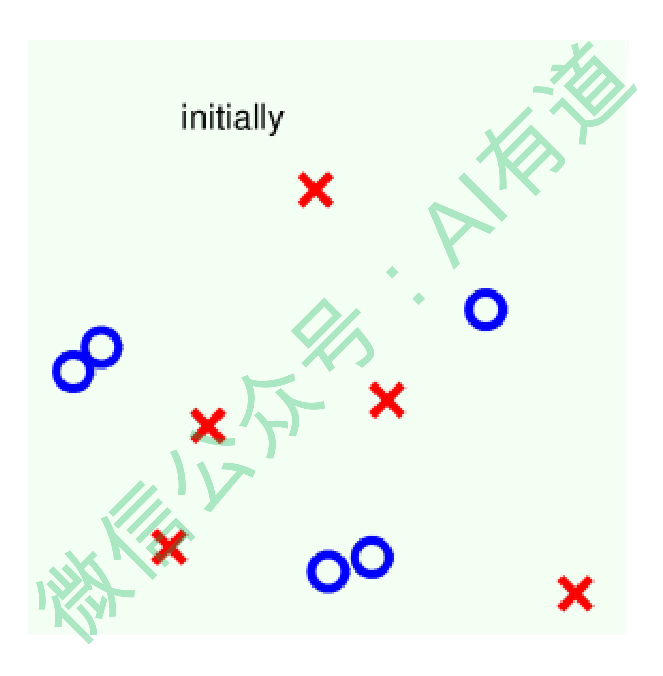


boosting view of AdaBoost:
if $\mathcal{A}$ is weak but always **slightly better than random** ($\epsilon_t \le \epsilon < \frac{1}{2}$), then (AdaBoost+$\mathcal{A}$) can be strong ($E_{\text{in}} = 0$ and E_{out} small)

Adaptive Boosting in Action

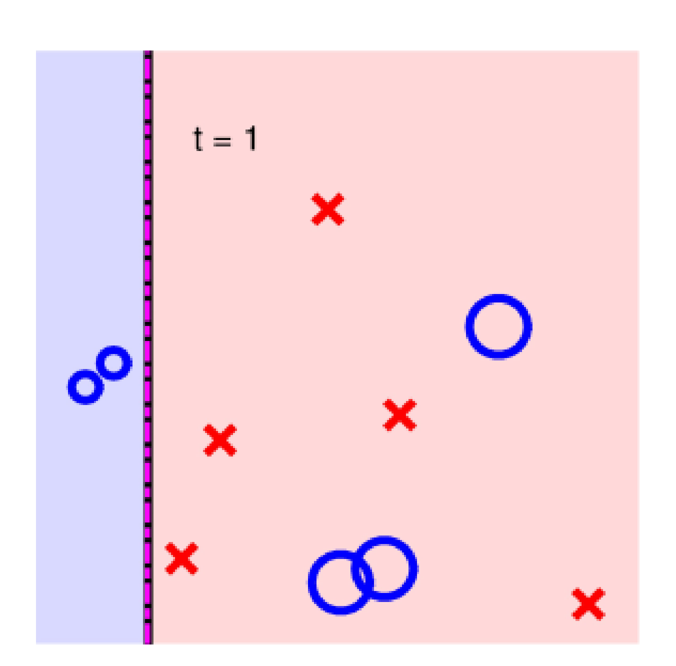
上一小节我们已经介绍了选择一个“弱弱”的算法A（$\epsilon_t \le \epsilon < \frac{1}{2}$，比乱猜好就行），就能经过多次迭代得到$E_{in} = 0$。我们称这种形式为decision stump模型。下面介绍一个例子，来看看AdaBoost是如何使用decision stump解决实际问题的。

如下图所示，二维平面上分布一些正负样本点，利用decision stump来做切割。

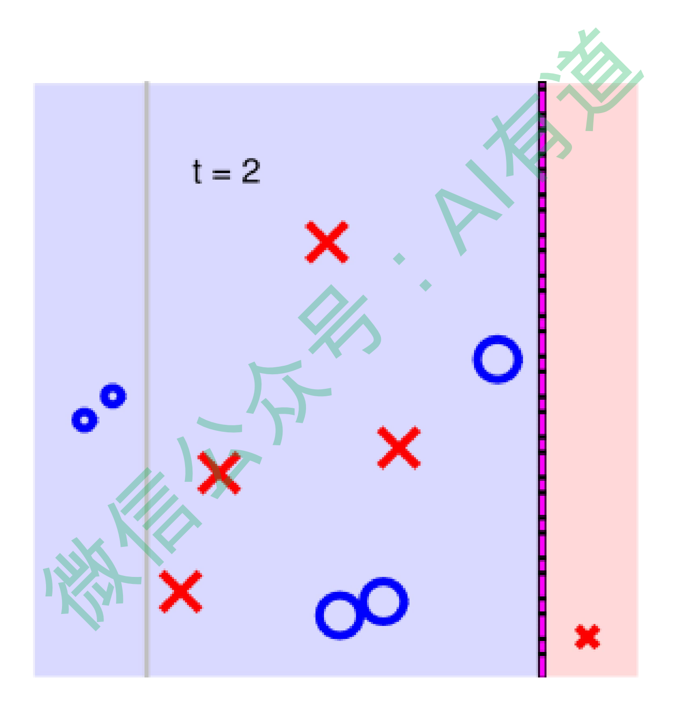


第一步：



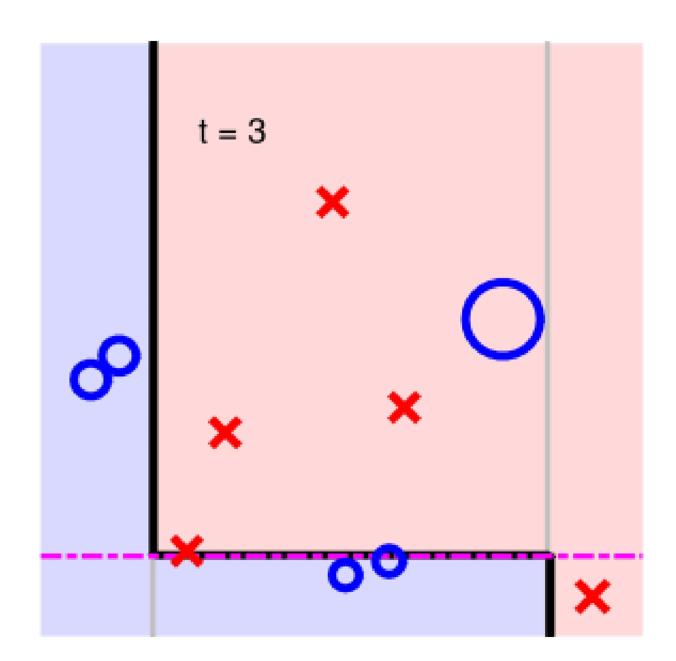


第二步:

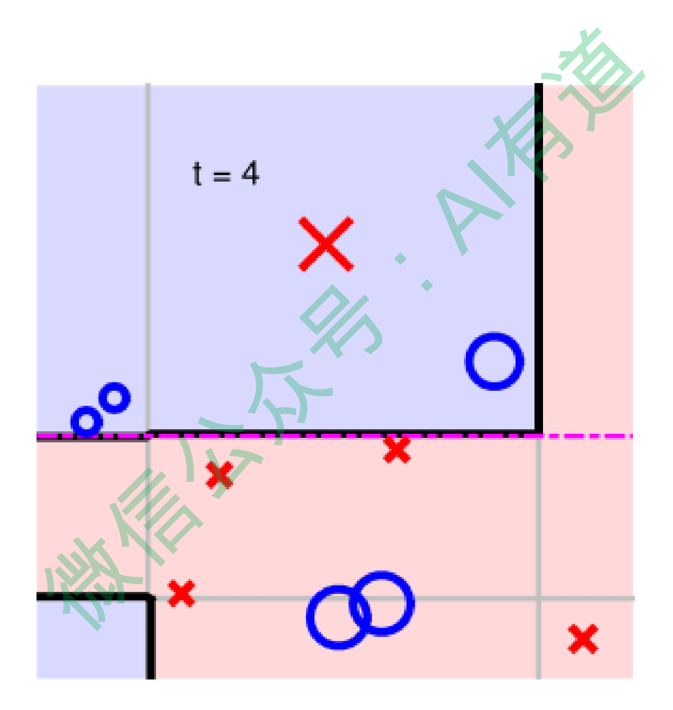


第三步:



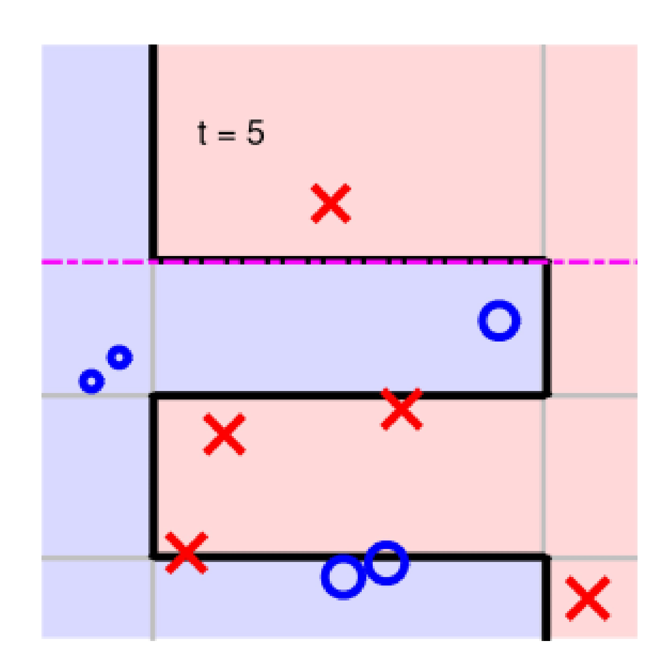


第四步:

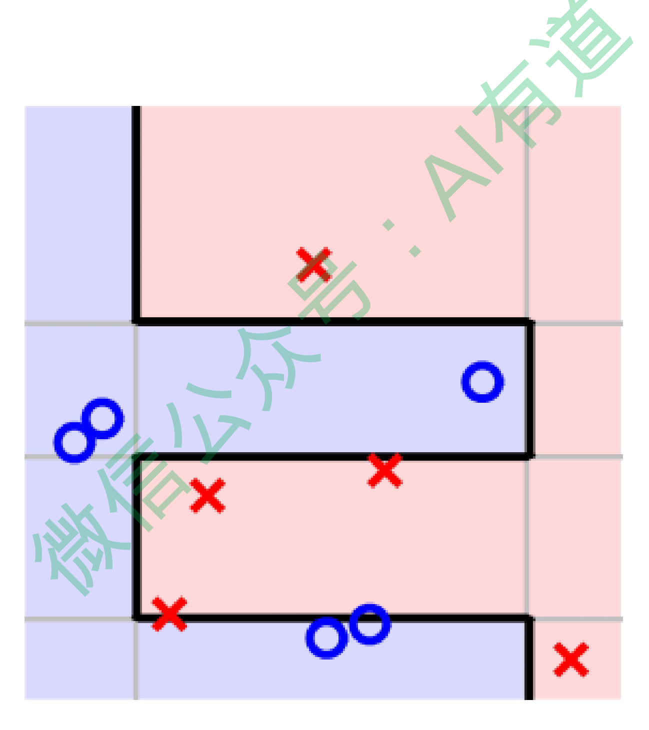


第五步:



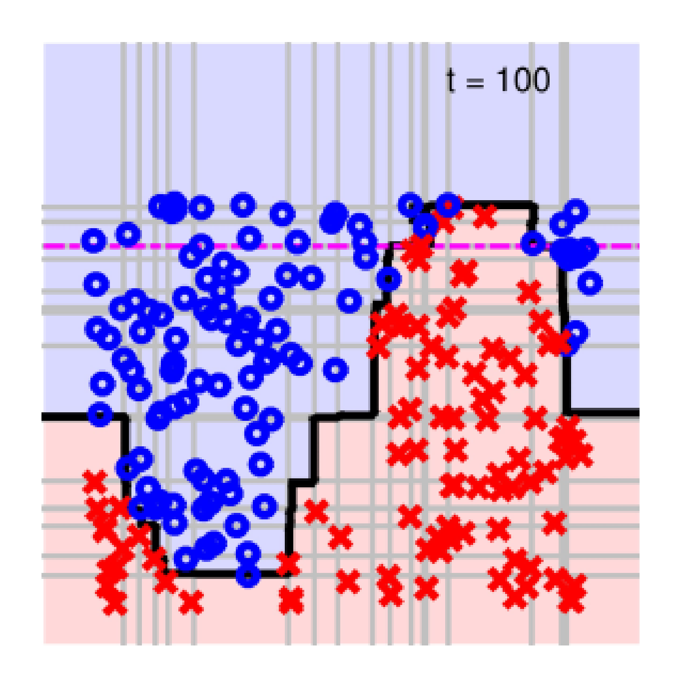


可以看到，经过5次迭代之后，所有的正负点已经被完全分开了，则最终得到的分类线为：



另外一个例子，对于一个相对比较复杂的数据集，如下图所示。它的分界线从视觉上看应该是一个sin波的形式。如果我们再使用AdaBoost算法，通过decision stump来做切割。在迭代切割100次后，得到的分界线如下所示。





可以看出，AdaBoost-Stump这种非线性模型得到的分界线对正负样本有较好的分离效果。

课程中还介绍了一个AdaBoost-Stump在人脸识别方面的应用：



original picture by F.U.S.I.A. assistant and derivative work by Sylenius via Wikimedia Commons

The World's First 'Real-Time' Face Detection Program

- AdaBoost-Stump as core model: linear aggregation of key patches selected out of 162,336 possibilities in 24x24 images
  —**feature selection** achieved through AdaBoost-Stump
- modified linear aggregation *G* to rule out non-face earlier
  —**efficiency** achieved through modified linear aggregation

总结

本节课主要介绍了Adaptive Boosting。首先通过讲一个老师教小学生识别苹果的例子，来引入Boosting的思想，即把许多“弱弱”的hypotheses合并起来，变成很强的预测模型。然后重点介绍这种算法如何实现，关键在于每次迭代时，给予样本不同的系数u，宗旨是放大错误样本，缩小正确样本，得到不同的小矩g。并且在每次迭代时根



据错误ϵ值的大小，给予不同g_t不同的权重。最终由不同的g_t进行组合得到整体的预测模型G。实际证明，Adaptive Boosting能够得到有效的预测模型。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程





林轩田《机器学习技法》课程笔记9 -- Decision Tree

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了Adaptive Boosting。AdaBoost演算法通过调整每笔资料的权重，得到不同的hypotheses，然后将不同的hypothesis乘以不同的系数α进行线性组合。这种演算法的优点是，即使底层的演算法g不是特别好（只要比乱选好点），经过多次迭代后算法模型会越来越好，起到了boost提升的效果。本节课将在此基础上介绍一种新的aggregation算法：决策树（Decision Tree）。

Decision Tree Hypothesis

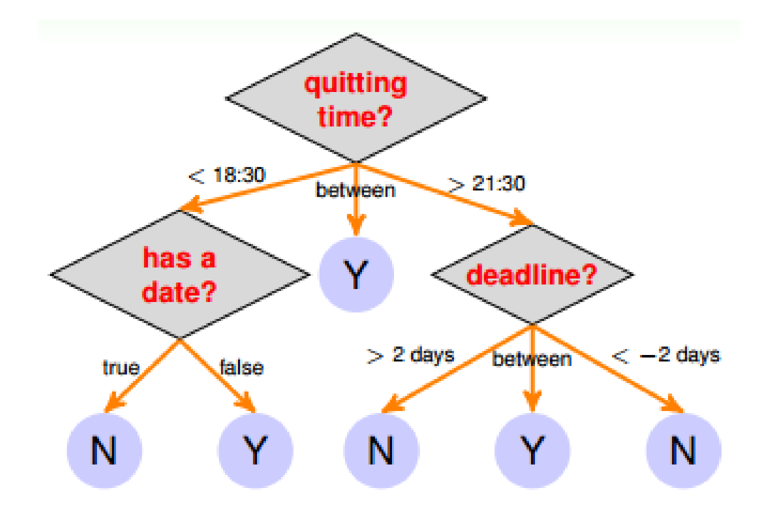
从第7节课开始，我们就一直在介绍aggregation model。aggregation的核心就是将许多可供选择使用的比较好的hypothesis融合起来，利用集体的智慧组合成G，使其得到更好的机器学习预测模型。下面，我们先来看看已经介绍过的aggregation type有哪些。

aggregation type	blending	learning
uniform	voting/averaging	Bagging
non-uniform	linear	AdaBoost
conditional	stacking	Decision Tree

aggregation type有三种：uniform，non-uniform，conditional。它有两种情况，一种是所有的g是已知的，即blending。对应的三种类型分别是voting/averaging，linear和stacking。另外一种情况是所有g未知，只能通过手上的资料重构g，即learning。其中uniform和non-uniform分别对应的是Bagging和AdaBoost算法，而conditional对应的就是我们本节课将要介绍的Decision Tree算法。

决策树（Decision Tree）模型是一种传统的算法，它的处理方式与人类思维十分相似。例如下面这个例子，对下班时间、约会情况、提交截止时间这些条件进行判断，从而决定是否要进行在线课程测试。如下图所示，整个流程类似一个树状结构。





图中每个条件和选择都决定了最终的结果，Y or N。蓝色的圆圈表示树的叶子，即最终的决定。

把这种树状结构对应到一个hypothesis G(x)中，G(x)的表达式为：

$$G(x)=\sum_{t=1}^{T}q_t(x)\cdot g_t(x)$$

G(x)由许多$g_t(x)$组成，即aggregation的做法。每个$g_t(x)$就代表上图中的蓝色圆圈（树的叶子）。这里的$g_t(x)$是常数，因为是处理简单的classification问题。我们把这些$g_t(x)$称为base hypothesis。$q_t(x)$表示每个$g_t(x)$成立的条件，代表上图中橘色箭头的部分。不同的$g_t(x)$对应于不同的$q_t(x)$，即从树的根部到顶端叶子的路径不同。图中中的菱形代表每个简单的节点。所以，这些base hypothesis和conditions就构成了整个G(x)的形式，就像一棵树一样，从根部到顶端所有的叶子都安全映射到上述公式上去了。

$$G(\mathbf{x})=\sum_{t=1}^{T}q_t(\mathbf{x})\cdot g_t(\mathbf{x})$$

- **base hypothesis** $g_t(\mathbf{x})$: leaf at end of path t, a **constant** here
- **condition** $q_t(\mathbf{x})$: [[is $\mathbf{x}$ on path t?]]
- usually with **simple internal nodes**

决策树实际上就是在模仿人类做决策的过程。一直以来，决策树的应用十分广泛而且



分类预测效果都很不错，而它在数学上的理论完备性不充分，倒也不必在意。

如果从另外一个方面来看决策树的形式，不同于上述G(x)的公式，我们可以利用条件分支的思想，将整体G(x)分成若干个$G_c(x)$，也就是把整个大树分成若干个小树，如下所示：

$$G(x)=\sum_{c=1}^{C}[b(x)=c]\cdot G_c(x)$$

上式中，G(x)表示完整的大树，即full-tree hypothesis，b(x)表示每个分支条件，即branching criteria，$G_c(x)$表示第c个分支下的子树，即sub-tree。这种结构被称为递归型的数据结构，即将大树分割成不同的小树，再将小树继续分割成更小的子树。所以，决策树可以分为两部分：root和sub-trees。

Recursive View

$$G(\mathbf{x})=\sum_{c=1}^{C}[\![b(\mathbf{x})=c]\!]\cdot G_c(\mathbf{x})$$

- $G(\mathbf{x})$: full-tree hypothesis
- $b(\mathbf{x})$: branching criteria
- $G_c(\mathbf{x})$: sub-tree hypothesis at the c-th branch

在详细推导决策树算法之前，我们先来看一看它的优点和缺点。首先，decision tree的优点有：

- **模型直观，便于理解，应用广泛**
- **算法简单，容易实现**
- **训练和预测时，效率较高**

然而，decision tree也有相应的缺点：

- **缺少足够的理论支持**
- **如何选择合适的树结构对初学者来说比较困惑**
- **决策树代表性的演算法比较少**



Usefulness

- human-explainable: widely used in business/medical data analysis
- simple: even freshmen can implement one :-)
- efficient in prediction and training

However......

- heuristic: mostly little theoretical explanations
- heuristics: 'heuristics selection' confusing to beginners
- arguably no single representative algorithm

Decision Tree Algorithm

我们可以用递归形式将decision tree表示出来，它的基本的算法可以写成：

$$G(\mathbf{x}) = \sum_{c=1}^{C} [\![b(\mathbf{x}) = c]\!] \, G_c(\mathbf{x})$$

这个Basic Decision Tree Algorithm的流程可以分成四个部分，首先学习设定划分不同分支的标准和条件是什么；接着将整体数据集D根据分支个数C和条件，划为不同分支下的子集Dc；然后对每个分支下的Dc进行训练，得到相应的机器学习模型Gc；最后将所有分支下的Gc合并到一起，组成大矩G(x)。但值得注意的是，这种递归的形式需要终止条件，否则程序将一直进行下去。当满足递归的终止条件之后，将会返回基本的hypothesis $g_t(x)$。

function DecisionTree(data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$)
if termination criteria met
  return base hypothesis $g_t(\mathbf{x})$
else
  1. learn branching criteria $b(\mathbf{x})$
  2. split $\mathcal{D}$ to C parts $\mathcal{D}_c = \{(\mathbf{x}_n, y_n) : b(\mathbf{x}_n) = c\}$
  3. build sub-tree $G_c \leftarrow$ DecisionTree($\mathcal{D}_c$)
  4. return $G(\mathbf{x}) = \sum_{c=1}^{C} [\![b(\mathbf{x}) = c]\!] \, G_c(\mathbf{x})$

所以，决策树的基本演算法包含了四个选择：

- **分支个数（number of branches）**



- **分支条件（branching criteria）**
- **终止条件（termination criteria）**
- **基本算法（base hypothesis）**

下面我们来介绍一种常用的决策树模型算法，叫做Classification and Regression Tree(C&RT)。C&RT算法有两个简单的设定，首先，分支的个数C=2，即二叉树（binary tree）的数据结构；然后，每个分支最后的$g_t(x)$（数的叶子）是一个常数。按照最小化E_{in}的目标，对于binary/multiclass classification(0/1 error)问题，看正类和负类哪个更多，$g_t(x)$取所占比例最多的那一类y_n；对于regression(squared error)问题，$g_t(x)$则取所有y_n的平均值。

two simple choices

- $C = 2$ (binary tree)
- $g_t(\mathbf{x}) = E_{in}$-optimal constant
  - binary/multiclass classification (0/1 error): majority of $\{y_n\}$
  - regression (squared error): average of $\{y_n\}$

对于决策树的基本演算法流程，C&RT还有一些简单的设定。首先，C&RT分支个数C=2，一般采用上节课介绍过的decision stump的方法进行数据切割。也就是每次在一个维度上，只对一个特征feature将数据一分为二，左子树和右子树，分别代表不同的类别。然而，怎么切割才能让数据划分得最好呢（error最小）？C&RT中使用纯净度purifying这个概念来选择最好的decision stump。purifying的核心思想就是每次切割都尽可能让左子树和右子树中同类样本占得比例最大或者y_n都很接近（regression），即错误率最小。比如说classifiacation问题中，如果左子树全是正样本，右子树全是负样本，那么它的纯净度就很大，说明该分支效果很好。

more simple choices

- simple internal node for $C = 2$: $\{1, 2\}$**-output decision stump**
- 'easier' sub-tree: branch by **purifying**

$$b(\mathbf{x}) = \underset{\text{decision stumps } h(\mathbf{x})}{\operatorname{argmin}} \sum_{c=1}^{2} |\mathcal{D}_c \text{ with } h| \cdot \text{impurity}(\mathcal{D}_c \text{ with } h)$$

根据C&RT中purifying的思想，我们得到选择合适的分支条件b(x)的表达式如上所示。最好的decision stump重点包含两个方面：一个是刚刚介绍的分支纯净度purifying，



purifying越大越好，而这里使用purifying相反的概念impurity，则impurity越小越好；另外一个是左右分支纯净度所占的权重，权重大小由该分支的数据量决定，分支包含的样本个数越多，则所占权重越大，分支包含的样本个数越少，则所占权重越小。上式中的$|D_c\ with\ h|$代表了分支c所占的权重。这里b(x)类似于error function（这也是为什么使用impurity代替purifying的原因），选择最好的decision stump，让所有分支的不纯度最小化，使b(x)越小越好。

不纯度Impurity如何用函数的形式量化？一种简单的方法就是类比于E_{in}，看预测值与真实值的误差是多少。对于regression问题，它的impurity可表示为：

$$impurity(D)=\frac{1}{N}\sum_{n=1}^{N}(y_n-\overline{y})^2$$

其中，$\overline{y}$表示对应分支下所有y_n的均值。

对应classification问题，它的impurity可表示为：

$$impurity(D)=\frac{1}{N}\sum_{n=1}^{N}[y_n\neq y^*]$$

其中，y^*表示对应分支下所占比例最大的那一类。

by E_{in} of optimal constant

- regression error:

$$\text{impurity}(\mathcal{D})=\frac{1}{N}\sum_{n=1}^{N}(y_n-\bar{y})^2$$

with $\bar{y}$ = average of $\{y_n\}$

- classification error:

$$\text{impurity}(\mathcal{D})=\frac{1}{N}\sum_{n=1}^{N}[\![y_n\neq y^*]\!]$$

with y^* = majority of $\{y_n\}$

以上这些impurity是基于原来的regression error和classification error直接推导的。进一步来看classification的impurity functions，如果某分支条件下，让其中一个分支纯度最大，那么就选择对应的decision stump，即得到的classification error为：



$$1 - max_{1 \le k \le K} \frac{\sum_{n=1}^{N} [y_n = k]}{N}$$

其中，K为分支个数。

上面这个式子只考虑纯度最大的那个分支，更好的做法是将所有分支的纯度都考虑并计算在内，用基尼指数（Gini index）表示：

$$1 - \sum_{k=1}^{K} (\frac{\sum_{n=1}^{N} [y_n = k]}{N})^2$$

Gini index的优点是将所有的class在数据集中的分布状况和所占比例全都考虑了，这样让decision stump的选择更加准确。

for classification

- Gini index:

$$1 - \sum_{k=1}^{K} \left(\frac{\sum_{n=1}^{N} [\![y_n = k]\!]}{N} \right)^2$$

—all k considered together

- classification error:

$$1 - \max_{1 \le k \le K} \frac{\sum_{n=1}^{N} [\![y_n = k]\!]}{N}$$

—optimal $k = y^*$ only

对于决策树C&RT算法，通常来说，上面介绍的各种impurity functions中，Gini index更适合求解classification问题，而regression error更适合求解regression问题。

C&RT算法迭代终止条件有两种情况，第一种情况是当前各个分支下包含的所有样本y_n都是同类的，即不纯度impurity为0，表示该分支已经达到了最佳分类程度。第二种情况是该特征下所有的x_n相同，无法对其进行区分，表示没有decision stumps。遇到这两种情况，C&RT算法就会停止迭代。

'forced' to terminate when

- all y_n the same: impurity = 0 $\Longrightarrow g_t(\mathbf{x}) = y_n$
- all $\mathbf{x}_n$ the same: **no decision stumps**



所以，C&RT算法遇到迭代终止条件后就成为完全长成树（fully-grown tree）。它每次分支为二，是二叉树结构，采用purify来选择最佳的decision stump来划分，最终得到的叶子（$g_t(x)$）是常数。

Decision Tree Heuristics in C&RT

现在我们已经知道了C&RT算法的基本流程：

function DecisionTree(data $\mathcal{D} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$)
if cannot branch anymore
    return $g_t(\mathbf{x}) = E_{\text{in}}$-optimal constant
else
    ① learn branching criteria

$$b(\mathbf{x}) = \underset{\text{decision stumps } h(\mathbf{x})}{\text{argmin}} \sum_{c=1}^{2} |\mathcal{D}_c \text{ with } h| \cdot \text{impurity}(\mathcal{D}_c \text{ with } h)$$

    ② split $\mathcal{D}$ to 2 parts $\mathcal{D}_c = \{(\mathbf{x}_n, y_n) : b(\mathbf{x}_n) = c\}$
    ③ build sub-tree $G_c \leftarrow$ DecisionTree($\mathcal{D}_c$)
    ④ return $G(\mathbf{x}) = \sum_{c=1}^{2} [\![b(\mathbf{x}) = c]\!] \, G_c(\mathbf{x})$

可以看到C&RT算法在处理binary classification和regression问题时非常简单实用，而且，处理muti-class classification问题也十分容易。

考虑这样一个问题，有N个样本，如果我们每次只取一个样本点作为分支，那么在经过N-1次分支之后，所有的样本点都能完全分类正确。最终每片叶子上只有一个样本，有N片叶子，即必然能保证$E_{in} = 0$。这样看似是完美的分割，但是不可避免地造成VC Dimension无限大，造成模型复杂度增加，从而出现过拟合现象。为了避免overfit，我们需要在C&RT算法中引入正则化，来控制整个模型的复杂度。

考虑到避免模型过于复杂的方法是减少叶子（$g_t(x)$）的数量，那么可以令regularizer就为决策树中叶子的总数，记为$\Omega(G)$。正则化的目的是尽可能减少$\Omega(G)$的值。这样，regularized decision tree的形式就可以表示成：

$$argmin_{(all\ possible\ G)}\ E_{in}(G) + \lambda\Omega(G)$$

我们把这种regularized decision tree称为pruned decision tree。pruned是修剪的意思，通过regularization来修剪决策树，去掉多余的叶子，更简洁化，从而达到避免过



拟合的效果。

那么如何确定修剪多少叶子，修剪哪些叶子呢？假设由C&RT算法得到一棵完全长成树（fully-grown tree），总共10片叶子。首先分别减去其中一片叶子，剩下9片，将这10种情况比较，取E_{in}最小的那个模型；然后再从9片叶子的模型中分别减去一片，剩下8片，将这9种情况比较，取E_{in}最小的那个模型。以此类推，继续修建叶子。这样，最终得到包含不同叶子的几种模型，将这几个使用regularized decision tree的error function来进行选择，确定包含几片叶子的模型误差最小，就选择该模型。另外，参数λ可以通过validation来确定最佳值。

- need a **regularizer**, say, $\Omega(G) = \text{NumberOfLeaves}(G)$
- want **regularized** decision tree:

$$\underset{\text{all possible } G}{\text{argmin}} \; E_{\text{in}}(G) + \lambda\Omega(G)$$

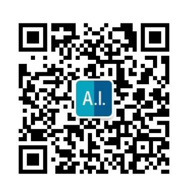
—called **pruned** decision tree

- cannot enumerate all possible G computationally:
  —often consider only
  - $G^{(0)}$ = fully-grown tree
  - $G^{(i)} = \text{argmin}_G \, E_{\text{in}}(G)$ such that G is **one-leaf removed** from $G^{(i-1)}$

我们一直讨论决策树上的叶子（features）都是numerical features，而实际应用中，决策树的特征值可能不是数字量，而是类别（categorical features）。对于numerical features，我们直接使用decision stump进行数值切割；而对于categorical features，我们仍然可以使用decision subset，对不同类别进行“左”和“右”，即是与不是（0和1）的划分。numerical features和categorical features的具体区别如下图所示：

numerical features	categorical features
blood pressure: 130, 98, 115, 147, 120	major symptom: fever, pain, tired, sweaty

branching for numerical	branching for categorical
decision stump	decision subset
$b(\mathbf{x}) = [\![x_i \leq \theta]\!] + 1$	$b(\mathbf{x}) = [\![x_i \in S]\!] + 1$
with $\theta \in \mathbb{R}$	with $S \subset \{1, 2, \ldots, K\}$



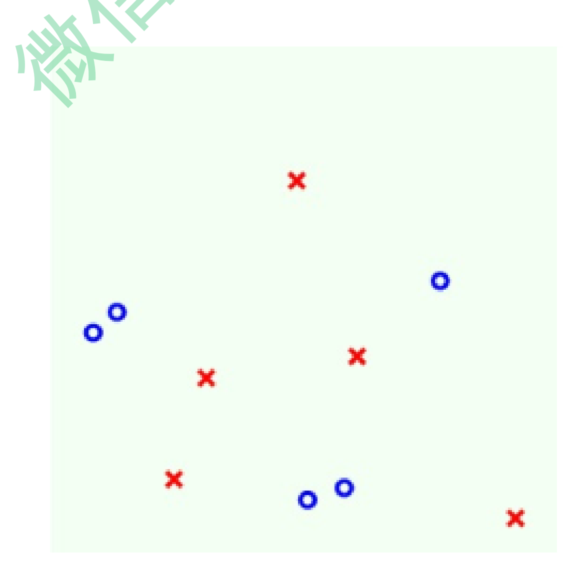
在决策树中预测中，还会遇到一种问题，就是当某些特征缺失的时候，没有办法进行切割和分支选择。一种常用的方法就是surrogate branch，即寻找与该特征相似的替代feature。如何确定是相似的feature呢？做法是在决策树训练的时候，找出与该特征相似的feature，如果替代的feature与原feature切割的方式和结果是类似的，那么就表明二者是相似的，就把该替代的feature也存储下来。当预测时遇到原feature缺失的情况，就用替代feature进行分支判断和选择。

if weight missing during prediction:

- what would human do?
  - go get weight
  - or, use threshold on height instead, because threshold on height $\approx$ threshold on weight
- surrogate branch:
  - maintain surrogate branch $b_1(\mathbf{x}), b_2(\mathbf{x}), \ldots \approx$ best branch $b(\mathbf{x})$ during training
  - allow missing feature for $b(\mathbf{x})$ during prediction by using surrogate instead

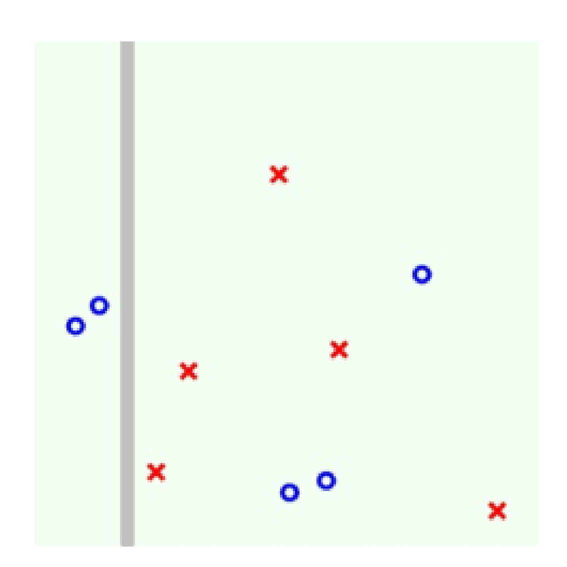
Decision Tree in Action

最后我们来举个例子看看C&RT算法究竟是如何进行计算的。例如下图二维平面上分布着许多正负样本，我们使用C&RT算法来对其进行决策树的分类。

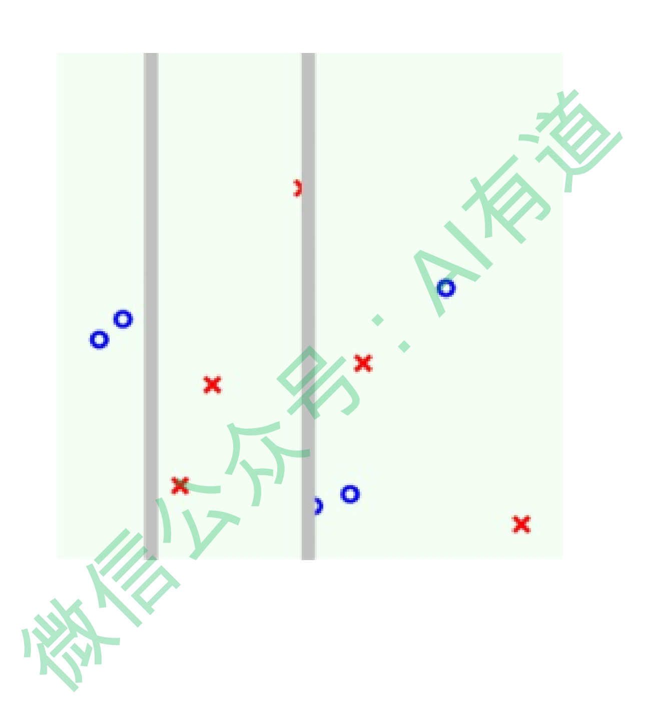


第一步：

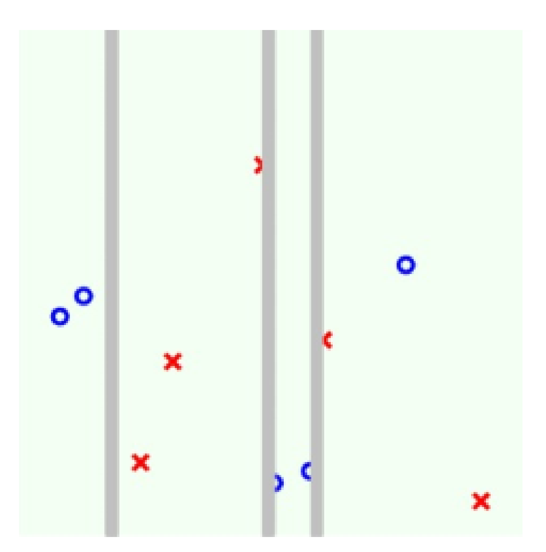




第二步：

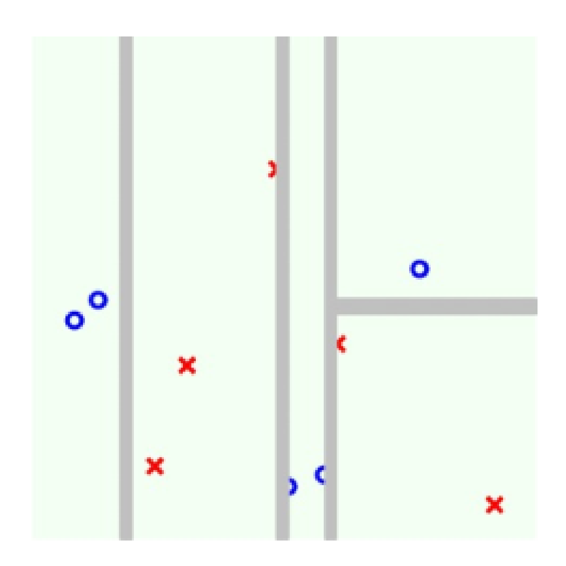


第三步：

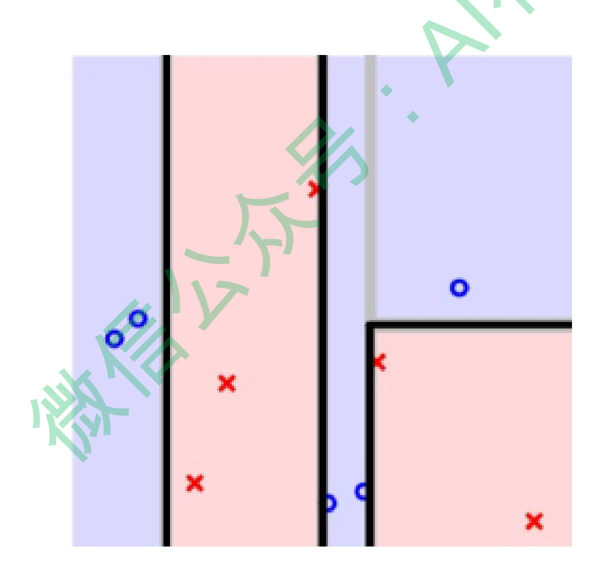


第四步：



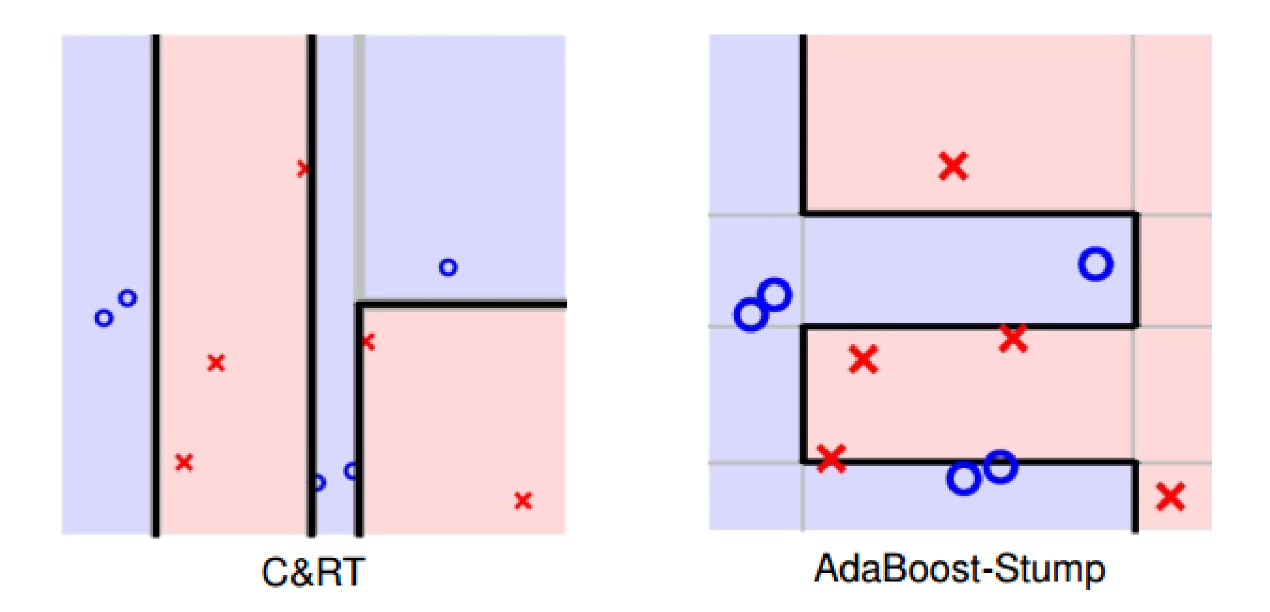


在进行第四步切割之后，我们发现每个分支都已经非常纯净了，没有办法继续往下切割。此时表明已经满足了迭代终止条件，这时候就可以回传base hypothesis，构成sub tree，然后每个sub tree再往上整合形成tree，最后形成我们需要的完全决策树。如果将边界添加上去，可得到下图：



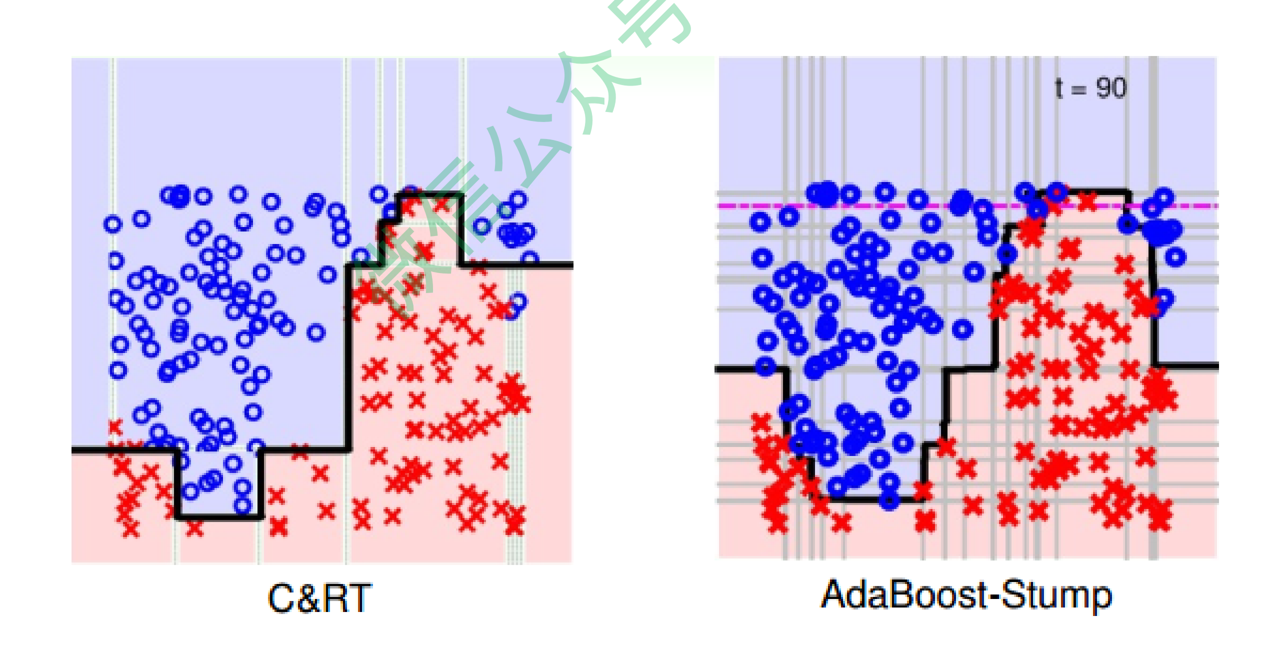
得到C&RT算法的切割方式之后，我们与AdaBoost-Stump算法进行比较：





我们之前就介绍过，AdaBoost-Stump算法的切割线是横跨整个平面的；而C&RT算法的切割线是基于某个条件的，所以一般不会横跨整个平面。比较起来，虽然C&RT和AdaBoost-Stump都采用decision stump方式进行切割，但是二者在细节上还是有所区别。

再看一个数据集分布比较复杂的例子，C&RT和AdaBoost-Stump的切割方式对比效果如下图所示：



通常来说，由于C&RT是基于条件进行切割的，所以C&RT比AdaBoost-Stump分类切割更有效率。总结一下，C&RT决策树有以下特点：



- human-explainable
- multiclass easily
- categorical features easily
- missing features easily
- efficient non-linear training (and testing)

—almost no other learning model share all such specialties, except for other decision trees

总结：

本节课主要介绍了Decision Tree。首先将decision tree hypothesis对应到不同分支下的矩$g_t(x)$。然后再介绍决策树算法是如何通过递归的形式建立起来。接着详细研究了决策树C&RT算法对应的数学模型和算法架构流程。最后通过一个实际的例子来演示决策树C&RT算法是如何一步一步进行分类的。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程



林轩田《机器学习技法》课程笔记10 -- Random Forest

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了Decision Tree模型。Decision Tree算法的核心是通过递归的方式，将数据集不断进行切割，得到子分支，最终形成数的结构。C&RT算法是决策树比较简单和常用的一种算法，其切割的标准是根据纯度来进行，每次切割都是为了让分支内部纯度最大。最终，决策树不同的分支得到不同的$g_t(x)$（即树的叶子，C&RT算法中，$g_t(x)$是常数）。本节课将介绍随机森林（Random Forest）算法，它是我们之前介绍的Bagging和上节课介绍的Decision Tree的结合。

Random Forest Algorithm

首先我们来复习一下之前介绍过的两个机器学习模型：Bagging和Decision Tree。Bagging是通过bootstrap的方式，从原始的数据集D中得到新的$\hat{D}$；然后再使用一些base algorithm对每个$\hat{D}$都得到相应的g_t；最后将所有的g_t通过投票uniform的形式组合成一个G，G即为我们最终得到的模型。Decision Tree是通过递归形式，利用分支条件，将原始数据集D切割成一个个子树结构，长成一棵完整的树形结构。Decision Tree最终得到的G(x)是由相应的分支条件b(x)和分支树$G_c(x)$递归组成。

Bagging

function Bag($\mathcal{D}, \mathcal{A}$)
For $t = 1, 2, \ldots, T$
1. request size-N' data $\tilde{\mathcal{D}}_t$ by bootstrapping with $\mathcal{D}$
2. obtain base g_t by $\mathcal{A}(\tilde{\mathcal{D}}_t)$

return $G = \text{Uniform}(\{g_t\})$

—reduces variance
by voting/averaging

Decision Tree

function DTree($\mathcal{D}$)
if termination return base g_t
else
1. learn $b(\mathbf{x})$ and split $\mathcal{D}$ to $\mathcal{D}_c$ by $b(\mathbf{x})$
2. build $G_c \leftarrow \text{DTree}(\mathcal{D}_c)$
3. return $G(\mathbf{x}) = \sum_{c=1}^{C} [\![b(\mathbf{x}) = c]\!] \, G_c(\mathbf{x})$

—large variance
especially if fully-grown

Bagging和Decison Tree算法各自有一个很重要的特点。Bagging具有减少不同g_t的



差variance的特点。这是因为Bagging采用投票的形式，将所有g_t uniform结合起来，起到了求平均的作用，从而降低variance。而Decision Tree具有增大不同g_t的方差variance的特点。这是因为Decision Tree每次切割的方式不同，而且分支包含的样本数在逐渐减少，所以它对不同的资料D会比较敏感一些，从而不同的D会得到比较大的variance。

所以说，Bagging能减小variance，而Decision Tree能增大variance。如果把两者结合起来，能否发挥各自的优势，起到优势互补的作用呢？这就是我们接下来将要讨论的aggregation of aggregation，即使用Bagging的方式把众多的Decision Tree进行uniform结合起来。这种算法就叫做随机森林（Random Forest），它将完全长成的C&RT决策树通过bagging的形式结合起来，最终得到一个庞大的决策模型。

random forest (RF) = bagging + fully-grown C&RT decision tree

Random Forest算法流程图如下所示：

function RandomForest($\mathcal{D}$)
For $t = 1, 2, \ldots, T$
1. request size-N' data $\tilde{\mathcal{D}}_t$ by bootstrapping with $\mathcal{D}$
2. obtain tree g_t by DTree($\tilde{\mathcal{D}}_t$)

return $G = \text{Uniform}(\{g_t\})$

function DTree($\mathcal{D}$)
if termination return base g_t
else
1. learn $b(\mathbf{x})$ and split $\mathcal{D}$ to $\mathcal{D}_c$ by $b(\mathbf{x})$
2. build $G_c \leftarrow \text{DTree}(\mathcal{D}_c)$
3. return $G(\mathbf{x}) = \sum_{c=1}^{C} [\![b(\mathbf{x}) = c]\!]\, G_c(\mathbf{x})$

Random Forest算法的优点主要有三个。第一，不同决策树可以由不同主机并行训练生成，效率很高；第二，随机森林算法继承了C&RT的优点；第三，将所有的决策树通过bagging的形式结合起来，避免了单个决策树造成过拟合的问题。

- highly parallel/efficient to learn
- inherit pros of C&RT
- eliminate cons of fully-grown tree

以上是基本的Random Forest算法，我们再来看一下如何让Random Forest中决策树



的结构更有多样性。Bagging中，通过bootstrap的方法得到不同于D的D'，使用这些随机抽取的资料得到不同的g_t。除了随机抽取资料获得不同g_t的方式之外，还有另外一种方法，就是随机抽取一部分特征。例如，原来有100个特征，现在只从中随机选取30个来构成决策树，那么每一轮得到的树都由不同的30个特征构成，每棵树都不一样。假设原来样本维度是d，则只选择其中的d'（d'小于d）个维度来建立决策树结构。这类似是一种从d维到d'维的特征转换，相当于是从高维到低维的投影，也就是说d'维z空间其实就是d维x空间的一个随机子空间（subspace）。通常情况下，d'远小于d，从而保证算法更有效率。Random Forest算法的作者建议在构建C&RT每个分支b(x)的时候，都可以重新选择子特征来训练，从而得到更具有多样性的决策树。

another possibility for **diversity**:

randomly **sample** d' **features** from $\mathbf{x}$

- when sampling index $i_1, i_2, \ldots, i_{d'}$: $\mathbf{\Phi}(\mathbf{x}) = (x_{i_1}, x_{i_2}, \ldots, x_{i_{d'}})$
- $\mathcal{Z} \in \mathbb{R}^{d'}$: a **random subspace** of $\mathcal{X} \in \mathbb{R}^{d}$
- often $d' \ll d$, efficient for large d
  —can be generally applied on other models
- original RF **re-sample new subspace for each $b(\mathbf{x})$ in C&RT**

所以说，这种增强的Random Forest算法增加了random-subspace。

RF = bagging + random-subspace C&RT

上面我们讲的是随机抽取特征，除此之外，还可以将现有的特征x，通过数组p进行线性组合，来保持多样性：

$$\phi_i(x) = p_i^T x$$

这种方法使每次分支得到的不再是单一的子特征集合，而是子特征的线性组合（权重不为1）。好比在二维平面上不止得到水平线和垂直线，也能得到各种斜线。这种做法使子特征选择更加多样性。值得注意的是，不同分支i下的p_i是不同的，而且向量p_i中大部分元素为零，因为我们选择的只是一部分特征，这是一种低维映射。



more **powerful** features for **diversity**: row i other than natural basis

- **projection** (combination) with random row $\mathbf{p}_i$ of P: $\phi_i(\mathbf{x}) = \mathbf{p}_i^T \mathbf{x}$
- often consider **low-dimensional** projection:
  only d'' **non-zero** components in $\mathbf{p}_i$
- includes **random subspace** as **special case**:
  $d'' = 1$ and $\mathbf{p}_i \in$ natural basis
- original RF consider d' random **low-dimensional projections for each** $b(\mathbf{x})$ in C&RT

所以，这里的Random Forest算法又有增强，由原来的random-subspace变成了random-combination。顺便提一下，这里的random-combination类似于perceptron模型。

RF = bagging + random-**combination** C&RT
—randomness everywhere!

Out-Of-Bag Estimate

上一部分我们已经介绍了Random Forest算法，而Random Forest算法重要的一点就是Bagging。接下来将继续探讨bagging中的bootstrap机制到底蕴含了哪些可以为我们所用的东西。

通过bootstrap得到新的样本集D'，再由D'训练不同的g_t。我们知道D'中包含了原样本集D中的一些样本，但也有些样本没有涵盖进去。如下表所示，不同的g_t下，红色的表示在$\hat{D}_t$中没有这些样本。例如对g_1来说，(x_2, y_2)和(x_3, y_4)没有包含进去，对g_2来说，(x_1, y_1)和(x_2, y_2)没有包含进去，等等。每个g_t中，红色表示的样本被称为out-of-bag(OOB) example。

	g_1	g_2	g_3	$\cdots$	g_T
$(\mathbf{x}_1, y_1)$	$\tilde{\mathcal{D}}_1$	$\star$	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
$(\mathbf{x}_2, y_2)$	$\star$	$\star$	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
$(\mathbf{x}_3, y_3)$	$\star$	$\tilde{\mathcal{D}}_2$	$\star$		$\tilde{\mathcal{D}}_T$
$\cdots$					
$(\mathbf{x}_N, y_N)$	$\tilde{\mathcal{D}}_1$	$\tilde{\mathcal{D}}_2$	$\star$		$\star$

首先，我们来计算OOB样本到底有多少。假设bootstrap的数量N'=N，那么某个样本



(x_n, y_n)是OOB的概率是：

$$(1-\frac{1}{N})^N = \frac{1}{(\frac{N}{N-1})^N} = \frac{1}{(1+\frac{1}{N-1})^N} \approx \frac{1}{e}$$

其中，e是自然对数，N是原样本集的数量。由上述推导可得，每个g_t中，OOB数目大约是$\frac{1}{e}N$，即大约有三分之一的样本没有在bootstrap中被抽到。

然后，我们将OOB与之前介绍的Validation进行对比：

OOB

	g_1	g_2	g_3	$\cdots$	g_T
$(\mathbf{x}_1, y_1)$	$\tilde{\mathcal{D}}_1$	★	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
$(\mathbf{x}_2, y_2)$	★	★	$\tilde{\mathcal{D}}_3$		$\tilde{\mathcal{D}}_T$
$(\mathbf{x}_3, y_3)$	★	$\tilde{\mathcal{D}}_2$	★		$\tilde{\mathcal{D}}_T$
$\cdots$					
$(\mathbf{x}_N, y_N)$	$\tilde{\mathcal{D}}_1$	★	★		★

Validation

g_1^-	g_2^-	$\cdots$	g_M^-
$\mathcal{D}_{\text{train}}$	$\mathcal{D}_{\text{train}}$		$\mathcal{D}_{\text{train}}$
$\mathcal{D}_{\text{val}}$	$\mathcal{D}_{\text{val}}$		$\mathcal{D}_{\text{val}}$
$\mathcal{D}_{\text{val}}$	$\mathcal{D}_{\text{val}}$		$\mathcal{D}_{\text{val}}$
$\mathcal{D}_{\text{train}}$	$\mathcal{D}_{\text{train}}$		$\mathcal{D}_{\text{train}}$

在Validation表格中，蓝色的D_{train}用来得到不同的g_m^-，而红色的D_{val}用来验证各自的g_m^-。D_{train}与D_{val}没有交集，一般D_{train}是D_{val}的数倍关系。再看左边的OOB表格，之前我们也介绍过，蓝色的部分用来得到不同的g_t，而红色的部分是OOB样本。而我们刚刚也推导过，红色部分大约占N的$\frac{1}{e}$。通过两个表格的比较，我们发现OOB样本类似于D_{val}，那么是否能使用OOB样本来验证g_t的好坏呢？答案是肯定的。但是，通常我们并不需要对单个g_t进行验证。因为我们更关心的是由许多g_t组合成的G，即使g_t表现不太好，只要G表现足够好就行了。那么问题就转化成了如何使用OOB来验证G的好坏。方法是先看每一个样本(x_n, y_n)是哪些g_t的OOB资料，然后计算其在这些g_t上的表现，最后将所有样本的表现求平均即可。例如，样本(x_N, y_N)是g_2，g_3，g_T的OOB，则可以计算(x_N, y_N)在$G_N^-(x)$上的表现为：

$$G_N^-(x) = average(g_2, g_3, g_T)$$

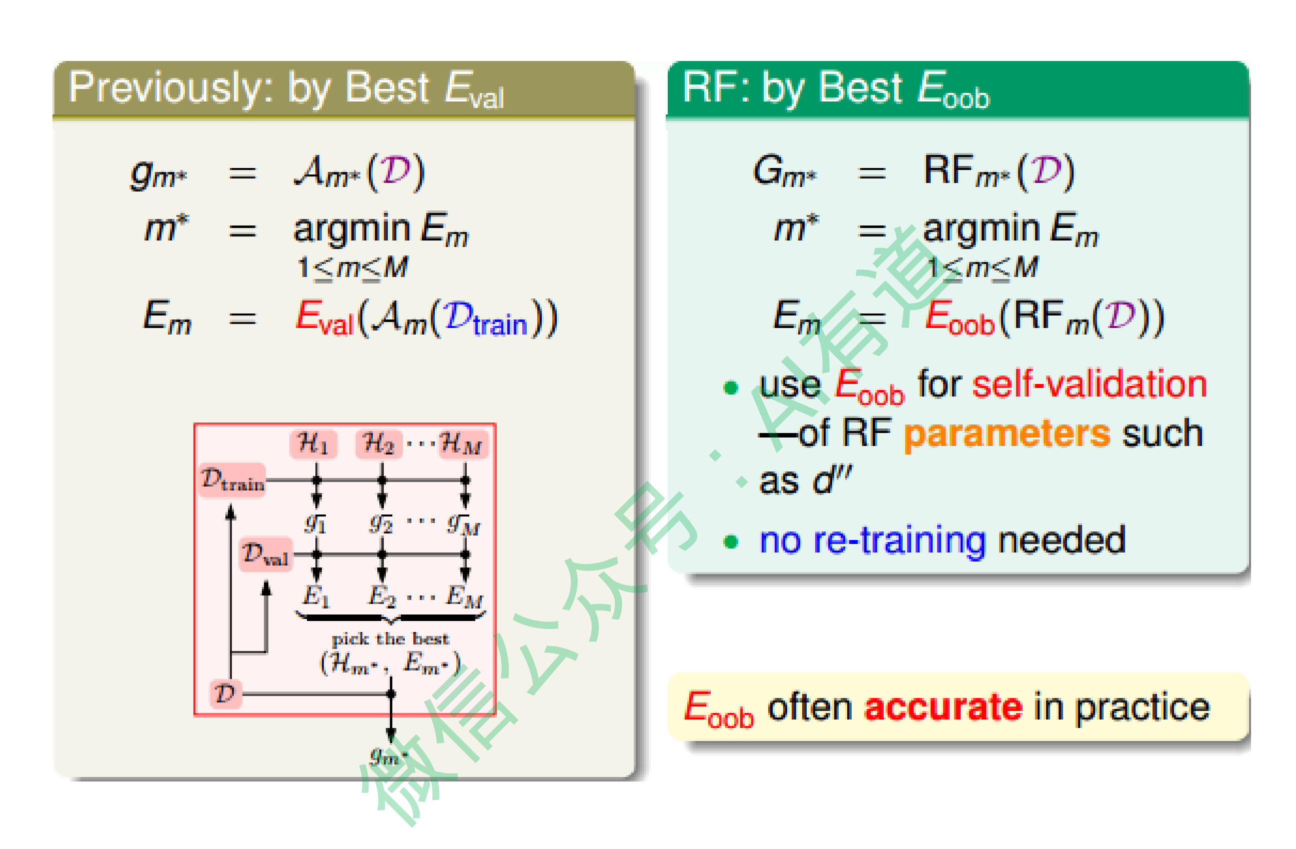
这种做法我们并不陌生，就像是我们之前介绍过的Leave-One-Out Cross Validation，每次只对一个样本进行g^-的验证一样，只不过这里选择的是每个样本是哪些g_t的OOB，然后再分别进行$G_n^-(x)$的验证。每个样本都当成验证资料一次（与留一法相同），最后计算所有样本的平均表现：

$$E_{oob}(G) = \frac{1}{N}\sum_{n=1}^{N} err(y_n, G_n^-(x_n))$$



$E_{oob}(G)$估算的就是G的表现好坏。我们把E_{oob}称为bagging或者Random Forest的self-validation。

这种self-validation相比于validation来说还有一个优点就是它不需要重复训练。如下图左边所示，在通过D_{val}选择到表现最好的$g^-_{m^*}$之后，还需要在D_{train}和D_{val}组成的所有样本集D上重新对该模型$g^-_{m^*}$训练一次，以得到最终的模型系数。但是self-validation在调整随机森林算法相关系数并得到最小的E_{oob}之后，就完成了整个模型的建立，无需重新训练模型。随机森林算法中，self-validation在衡量G的表现上通常相当准确。



Feature Selection

如果样本资料特征过多，假如有10000个特征，而我们只想从中选取300个特征，这时候就需要舍弃部分特征。通常来说，需要移除的特征分为两类：一类是冗余特征，即特征出现重复，例如“年龄”和“生日”；另一类是不相关特征，例如疾病预测的时候引入的“保险状况”。这种从d维特征到d'维特征的subset-transform $\Phi(x)$称为Feature Selection，最终使用这些d'维的特征进行模型训练。



for $\mathbf{x} = (x_1, x_2, \ldots, x_d)$, want to remove

- **redundant** features: like keeping one of 'age' and 'full birthday'
- **irrelevant** features: like insurance type for cancer prediction

and only 'learn' **subset-transform** $\Phi(\mathbf{x}) = (x_{i_1}, x_{i_2}, x_{i_{d'}})$ with $d' < d$ for $g(\Phi(\mathbf{x}))$

特征选择的优点是：

- **提高效率，特征越少，模型越简单**
- **正则化，防止特征过多出现过拟合**
- **去除无关特征，保留相关性大的特征，解释性强**

同时，特征选择的缺点是：

- **筛选特征的计算量较大**
- **不同特征组合，也容易发生过拟合**
- **容易选到无关特征，解释性差**

advantages:

- **efficiency**: simpler hypothesis and shorter prediction time
- **generalization**: 'feature noise' removed
- **interpretability**

disadvantages:

- **computation**: 'combinatorial' optimization in training
- **overfit**: 'combinatorial' selection
- **mis-interpretability**

值得一提的是，在decision tree中，我们使用的decision stump切割方式也是一种feature selection。

那么，如何对许多维特征进行筛选呢？我们可以通过计算出每个特征的重要性（即权重），然后再根据重要性的排序进行选择即可。

idea: if possible to calculate

$$\text{importance}(i) \text{ for } i = 1, 2, \ldots, d$$

then can select $i_1, i_2, \ldots, i_{d'}$ of top-d' importance



这种方法在线性模型中比较容易计算。因为线性模型的score是由每个特征经过加权求和而得到的，而加权系数的绝对值$|w_i|$正好代表了对应特征x_i的重要性为多少。$|w_i|$越大，表示对应特征x_i越重要，则该特征应该被选择。w的值可以通过对已有的数据集(x_i, y_i)建立线性模型而得到。

importance by linear model

$$\text{score} = \mathbf{w}^T\mathbf{x} = \sum_{i=1}^{d} w_i x_i$$

- intuitive estimate: importance(i) = $|w_i|$ with some 'good' **w**
- getting 'good' **w**: learned from data
- non-linear models? often **much harder**

然而，对于非线性模型来说，因为不同特征可能是非线性交叉在一起的，所以计算每个特征的重要性就变得比较复杂和困难。例如，Random Forest就是一个非线性模型，接下来，我们将讨论如何在RF下进行特征选择。

RF中，特征选择的核心思想是random test。random test的做法是对于某个特征，如果用另外一个随机值替代它之后的表现比之前更差，则表明该特征比较重要，所占的权重应该较大，不能用一个随机值替代。相反，如果随机值替代后的表现没有太大差别，则表明该特征不那么重要，可有可无。所以，通过比较某特征被随机值替代前后的表现，就能推断出该特征的权重和重要性。

那么random test中的随机值如何选择呢？通常有两种方法：一是使用uniform或者gaussian抽取随机值替换原特征；一是通过permutation的方式将原来的所有N个样本的第i个特征值重新打乱分布（相当于重新洗牌）。比较而言，第二种方法更加科学，保证了特征替代值与原特征的分布是近似的（只是重新洗牌而已）。这种方法叫做permutation test（随机排序测试），即在计算第i个特征的重要性的时候，将N个样本的第i个特征重新洗牌，然后比较D和$D^{(p)}$表现的差异性。如果差异很大，则表明第i个特征是重要的。



- which random values?
  - uniform, Gaussian, ...: $P(x_i)$ changed
  - bootstrap, **permutation** (of $\{x_{n,i}\}_{n=1}^N$): $P(x_i)$ approximately remained
- **permutation** test:

$$\text{importance}(i) = \text{performance}(\mathcal{D}) - \text{performance}(\mathcal{D}^{(p)})$$

with $\mathcal{D}^{(p)}$ is $\mathcal{D}$ with $\{x_{n,i}\}$ replaced by permuted $\{x_{n,i}\}_{n=1}^N$

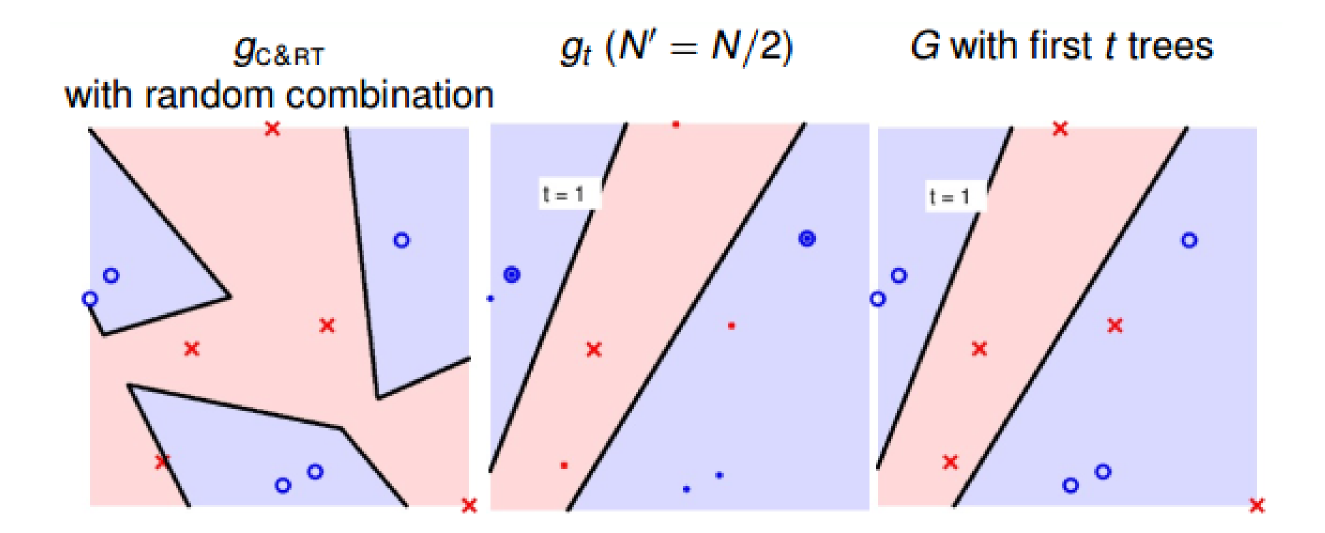
知道了permutation test的原理后，接下来要考虑的问题是如何衡量上图中的performance，即替换前后的表现。显然，我们前面介绍过performance可以用$E_{oob}(G)$来衡量。但是，对于N个样本的第i个特征值重新洗牌重置的$D^{(p)}$，要对它进行重新训练，而且每个特征都要重复训练，然后再与原D的表现进行比较，过程非常繁琐。为了简化运算，RF的作者提出了一种方法，就是把permutation的操作从原来的training上移到了OOB validation上去，记为$E_{oob}(G^{(p)}) \rightarrow E_{oob}^{(p)}(G)$。也就是说，在训练的时候仍然使用D，但是在OOB验证的时候，将所有的OOB样本的第i个特征重新洗牌，验证G的表现。这种做法大大简化了计算复杂度，在RF的feature selection中应用广泛。

- performance($\mathcal{D}^{(p)}$): needs re-training and validation in general
- **'escaping' validation**? **OOB** in RF
- original RF solution: importance$(i) = E_{\text{oob}}(G) - E_{\text{oob}}^{(p)}(G)$, where $E_{\text{oob}}^{(p)}$ comes from replacing each request of $x_{n,i}$ by a **permuted OOB** value

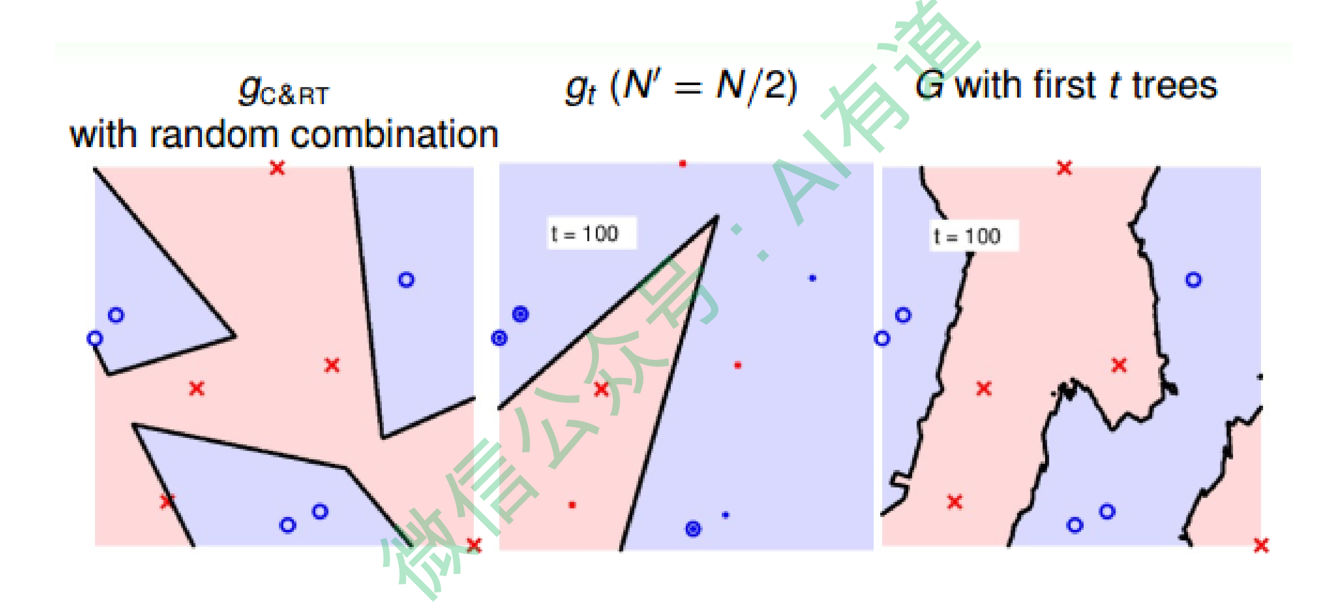
Random Forest in Action

最后，我们通过实际的例子来看一下RF的特点。首先，仍然是一个二元分类的例子。如下图所示，左边是一个C&RT树没有使用bootstrap得到的模型分类效果，其中不同特征之间进行了随机组合，所以有斜线作为分类线；中间是由bootstrap（N'=N/2）后生成的一棵决策树组成的随机森林，图中加粗的点表示被bootstrap选中的点；右边是将一棵决策树进行bagging后的分类模型，效果与中间图是一样的，都是一棵树。

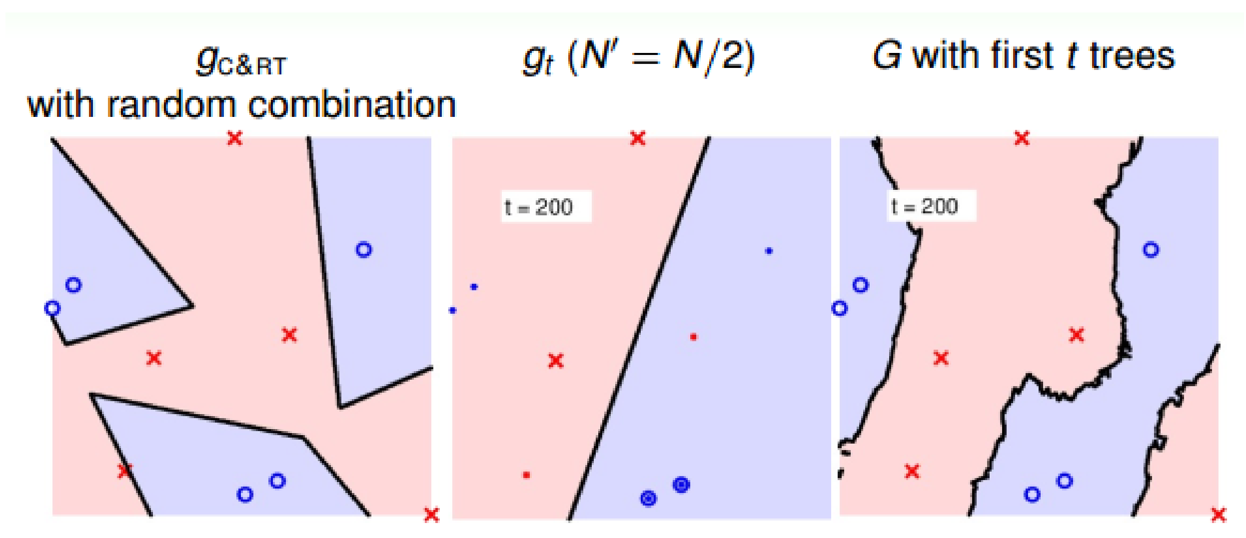




当t=100，即选择了100棵树时，中间的模型是第100棵决策树构成的，还是只有一棵树；右边的模型是由100棵决策树bagging起来的，如下图所示：

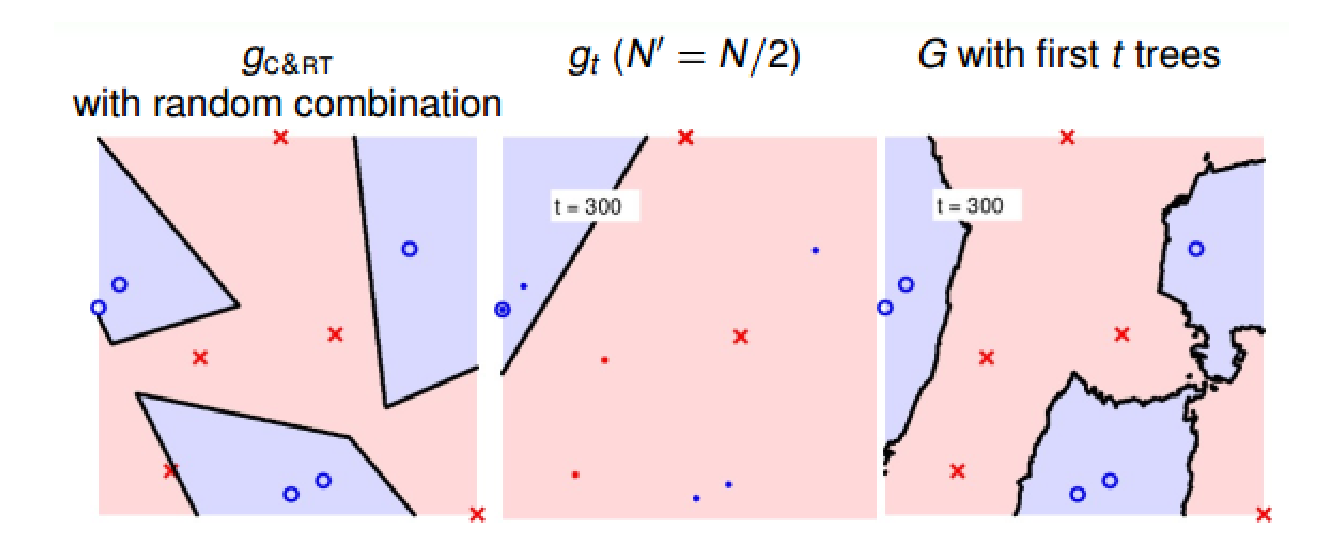


当t=200时：

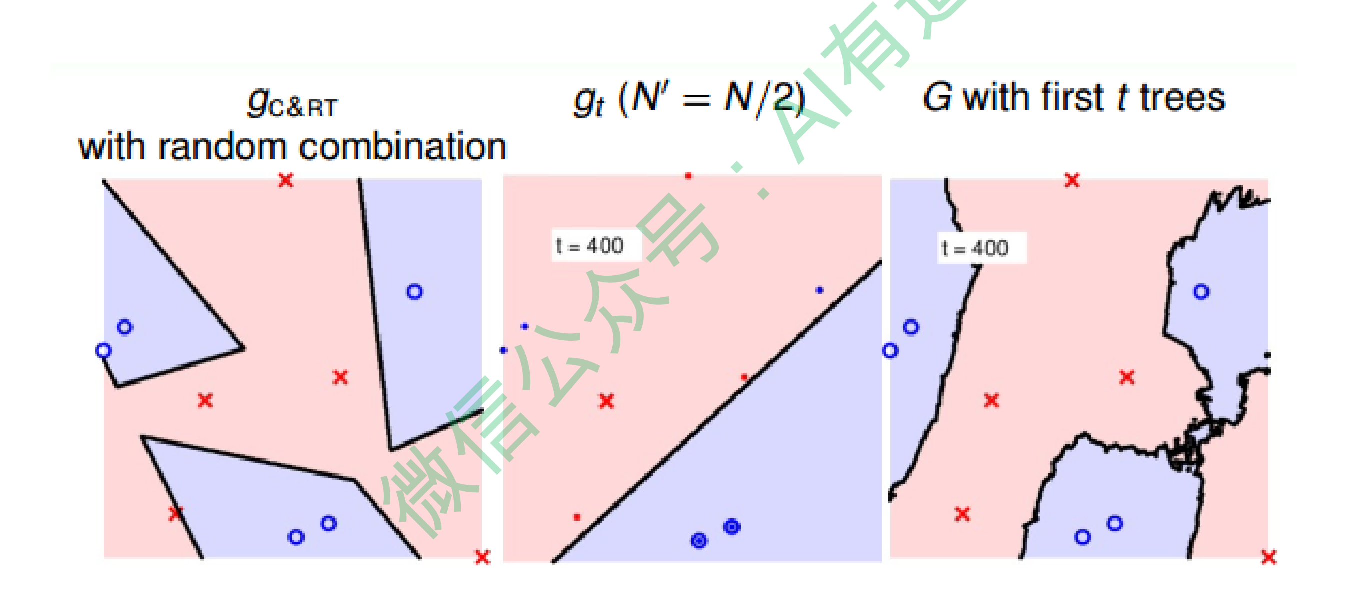




当t=300时:

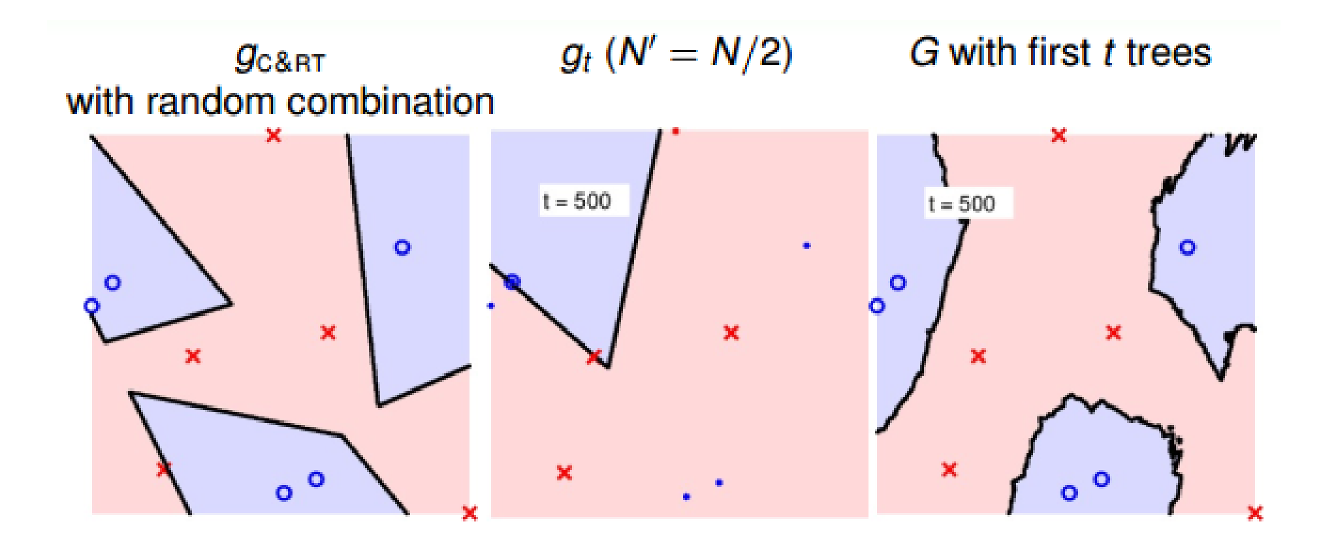



当t=400时:

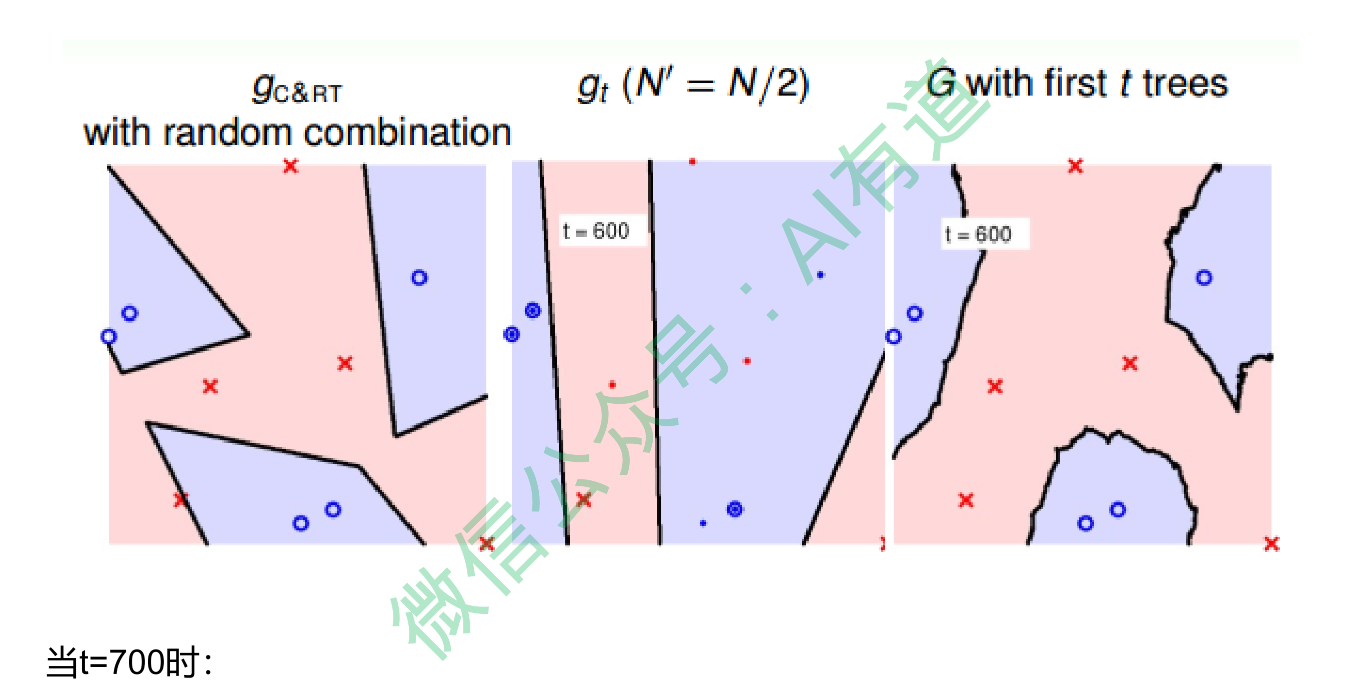


当t=500时:

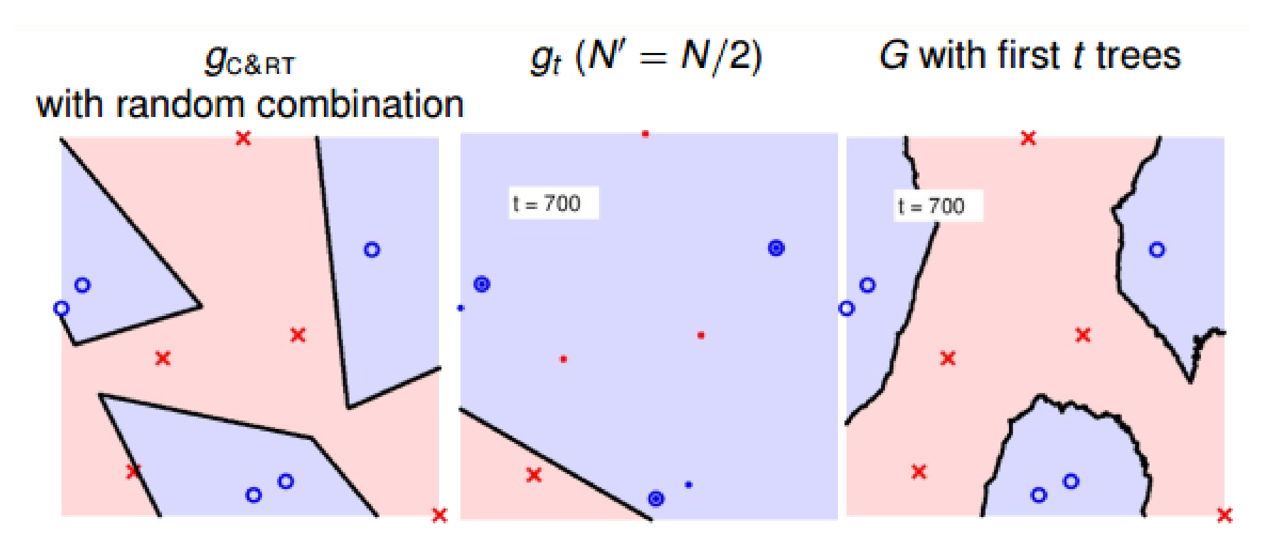




当t=600时：

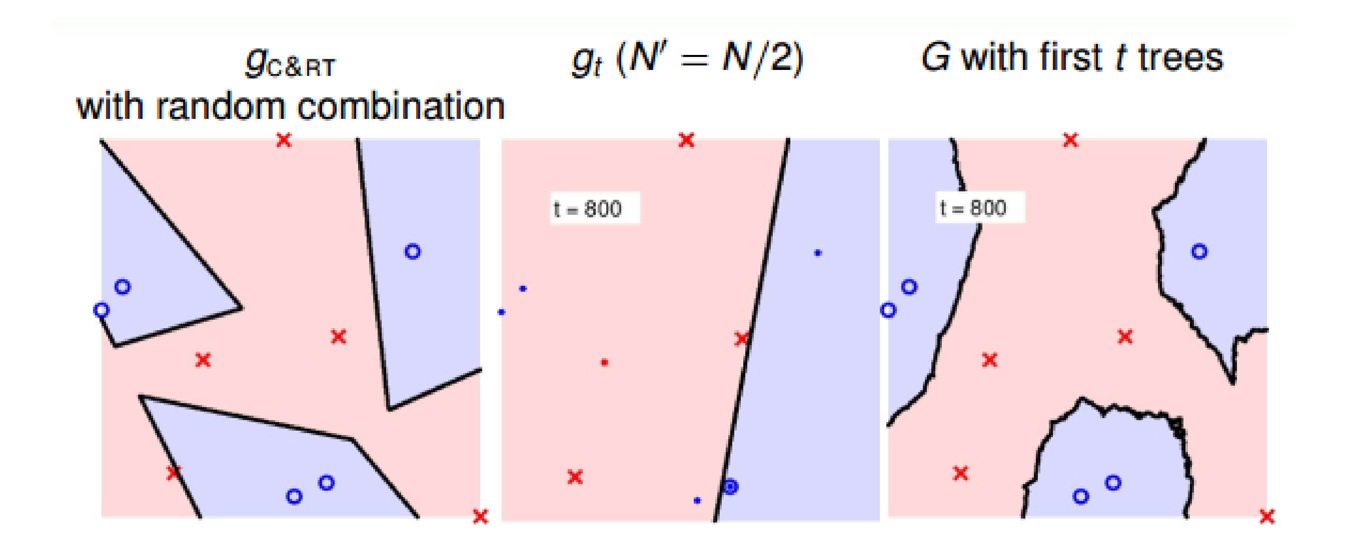


当t=700时：

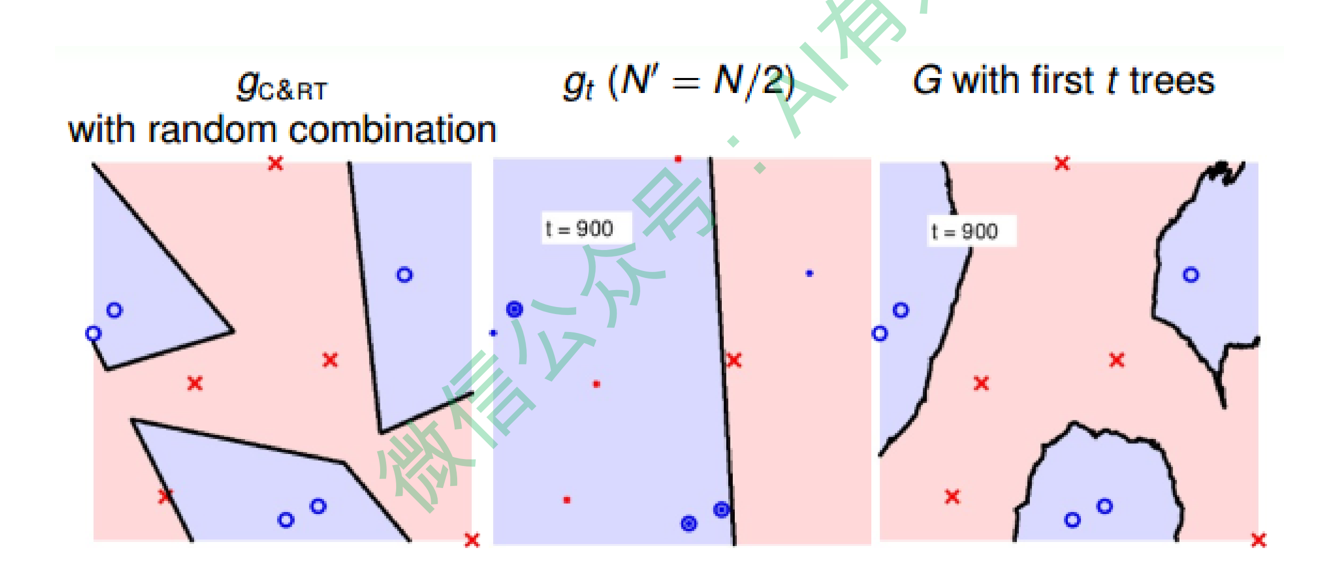




当t=800时:

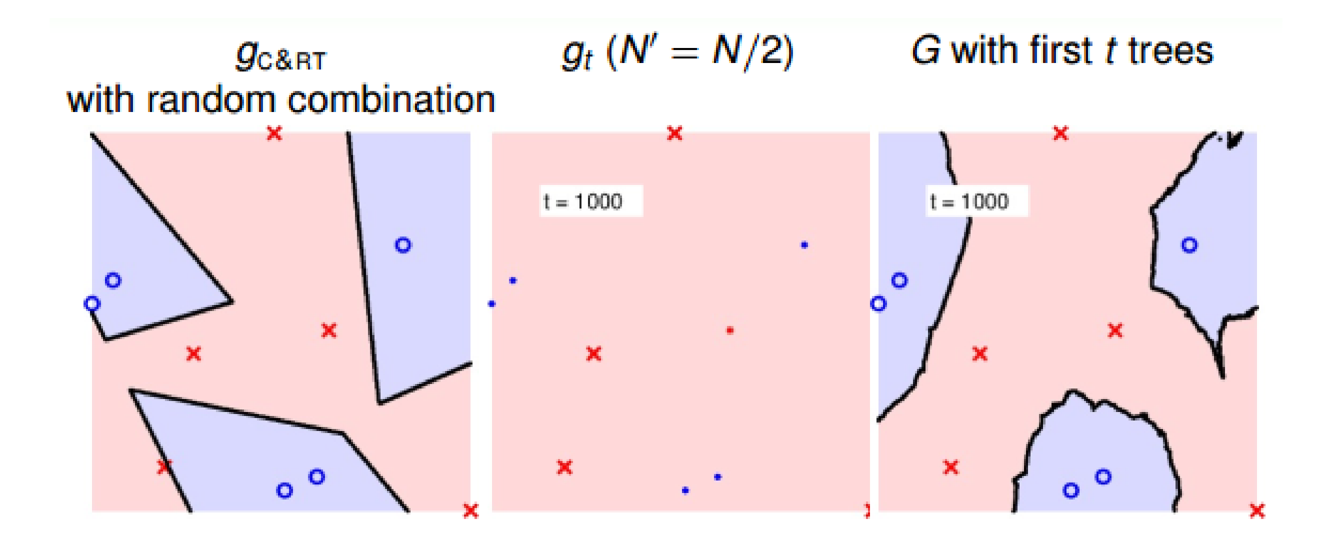



当t=900时:



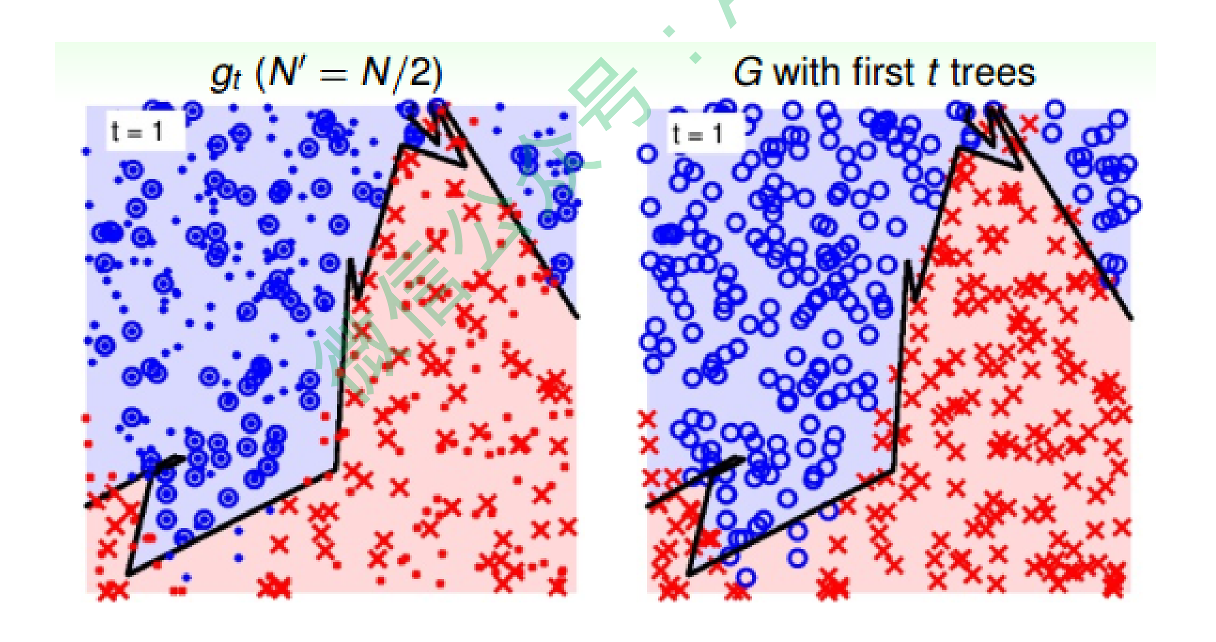
当t=1000时:





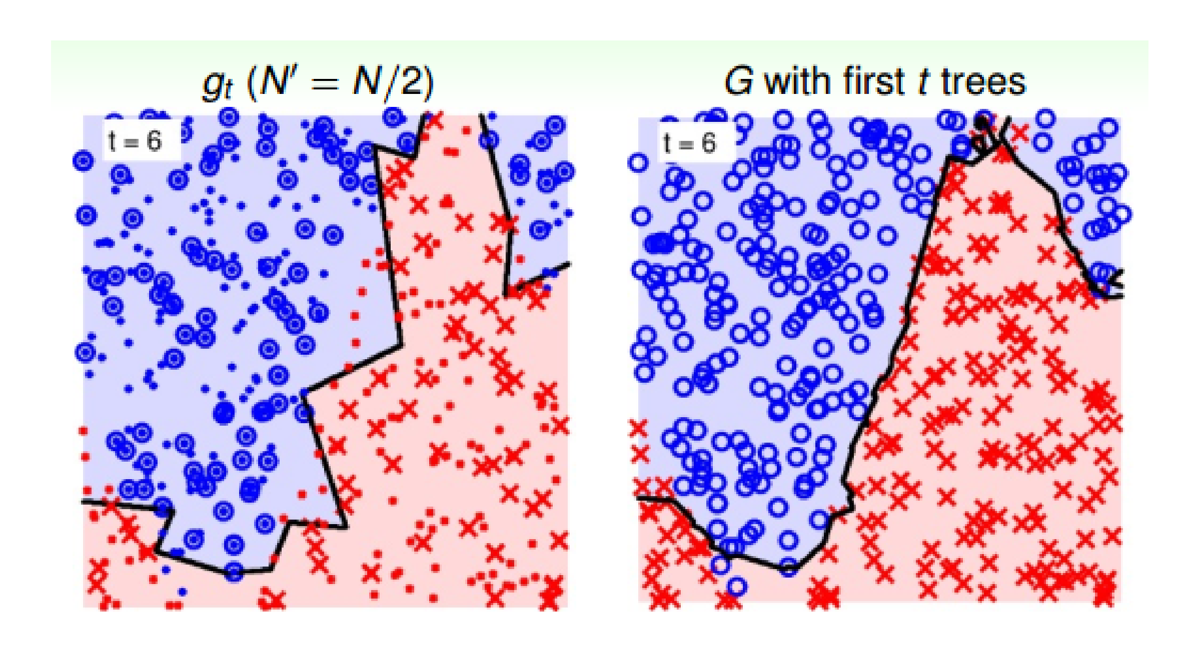
随着树木个数的增加，我们发现，分界线越来越光滑而且得到了large-margin-like boundary，类似于SVM一样的效果。也就是说，树木越多，分类器的置信区间越大。

然后，我们再来看一个比较复杂的例子，二维平面上分布着许多离散点，分界线形如sin函数。当只有一棵树的时候（t=1），下图左边表示单一树组成的RF，右边表示所有树bagging组合起来构成的RF。因为只有一棵树，所以左右两边效果一致。

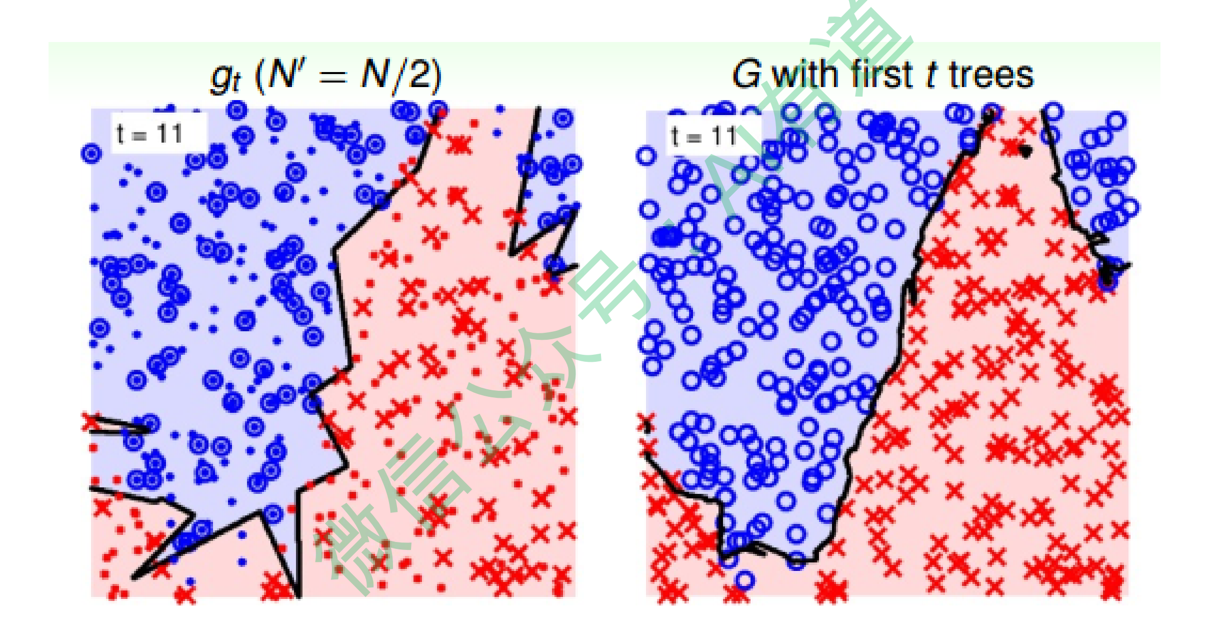


当t=6时：



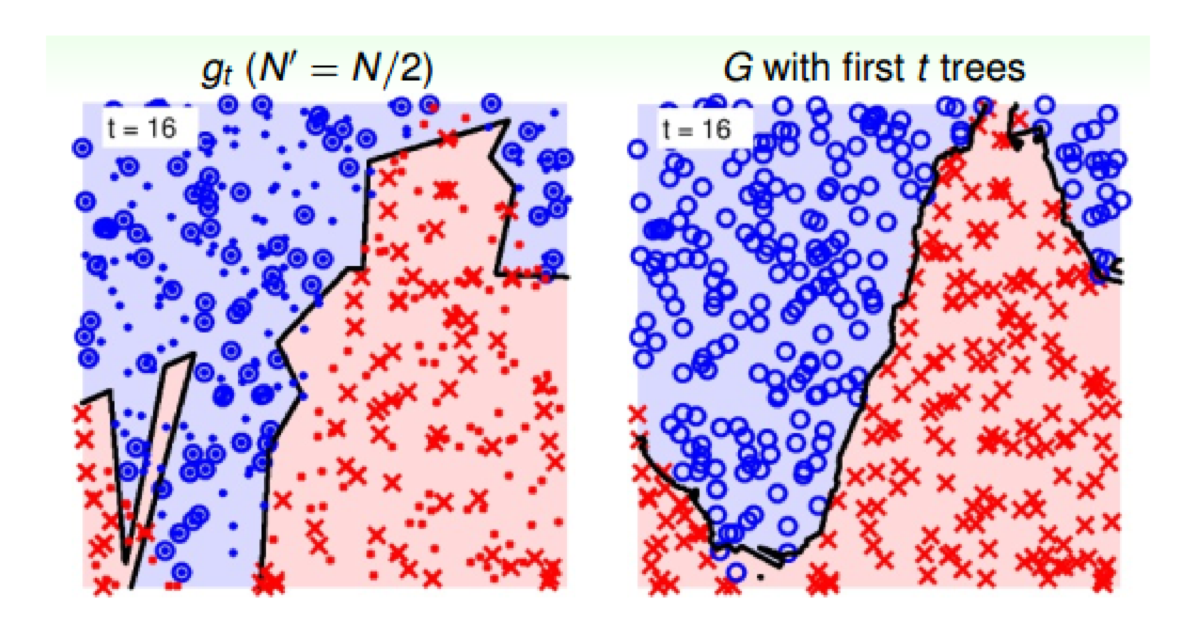

g_t $(N' = N/2)$
G with first t trees
t = 6
t = 6

当t=11时：

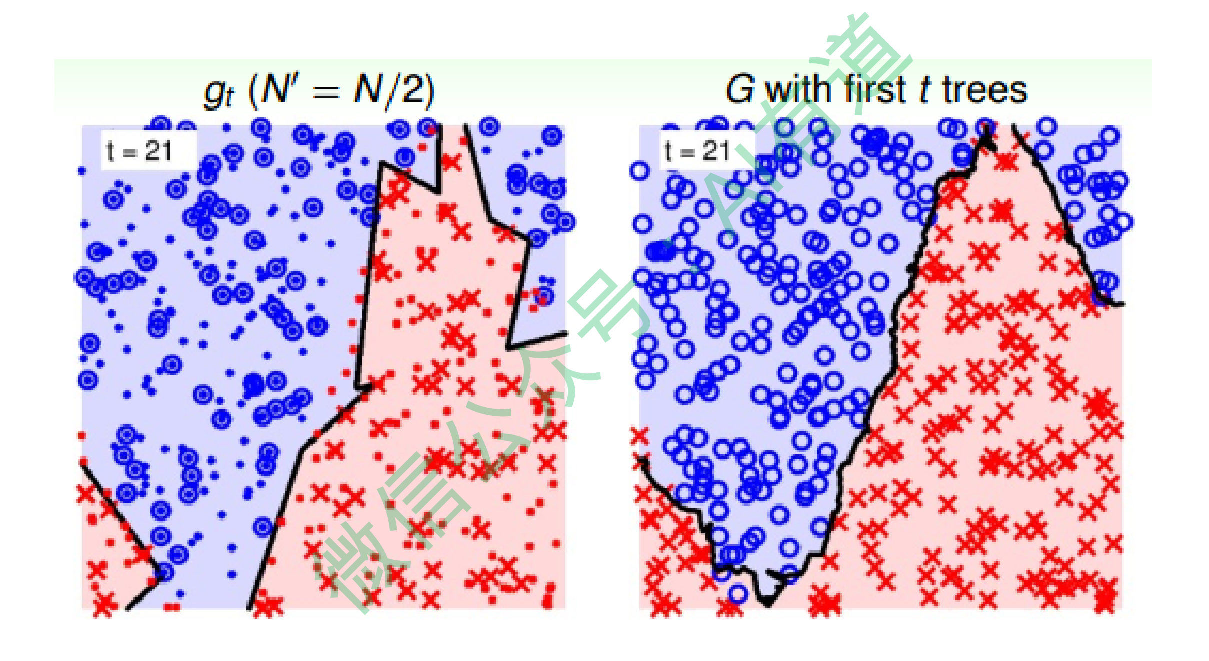

g_t $(N' = N/2)$
G with first t trees
t = 11
t = 11

当t=16时：





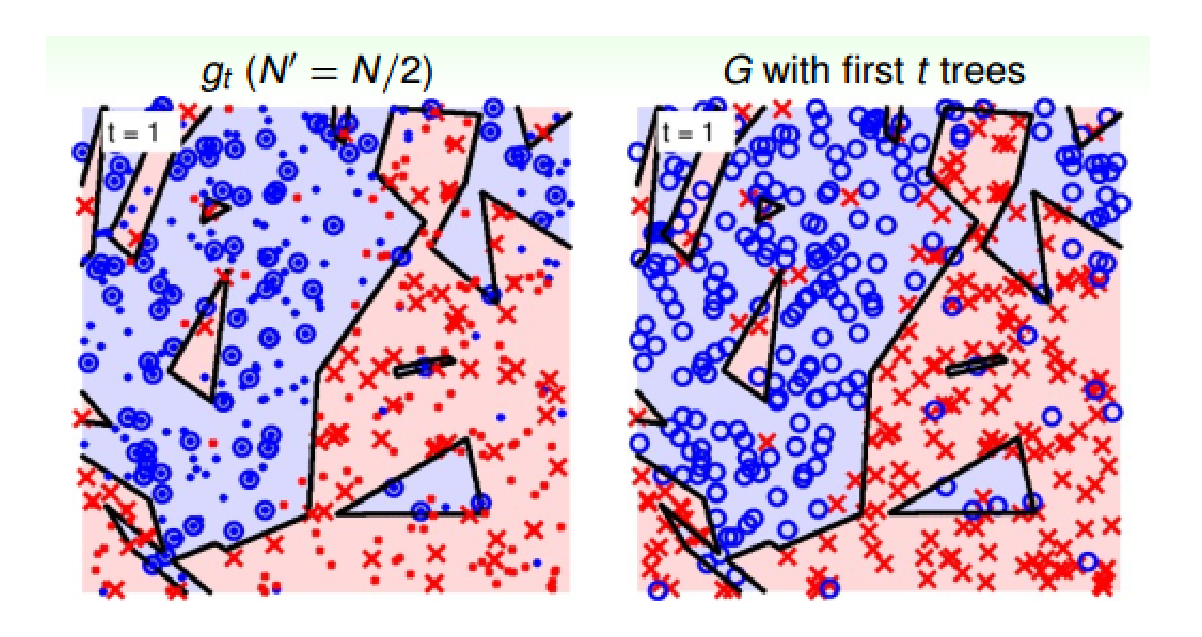
当t=21时：



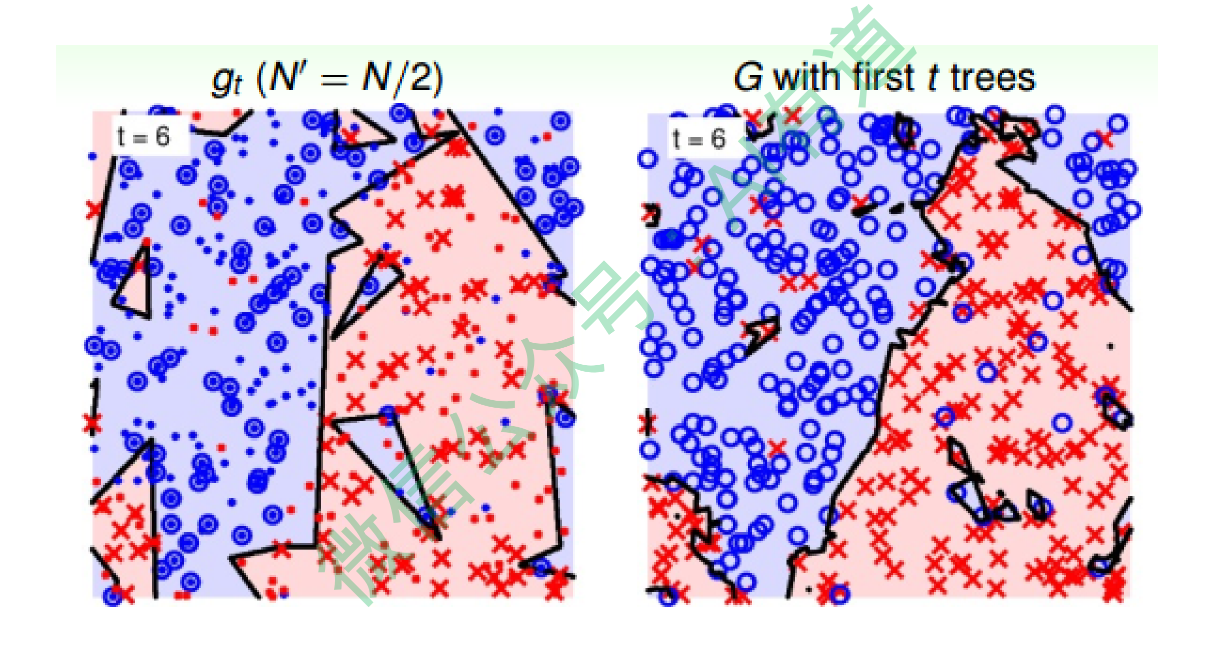
可以看到，当RF由21棵树构成的时候，分界线就比较平滑了，而且它的边界比单一树构成的RF要robust得多，更加平滑和稳定。

最后，基于上面的例子，再让问题复杂一点：在平面上添加一些随机噪声。当t=1时，如下图所示：



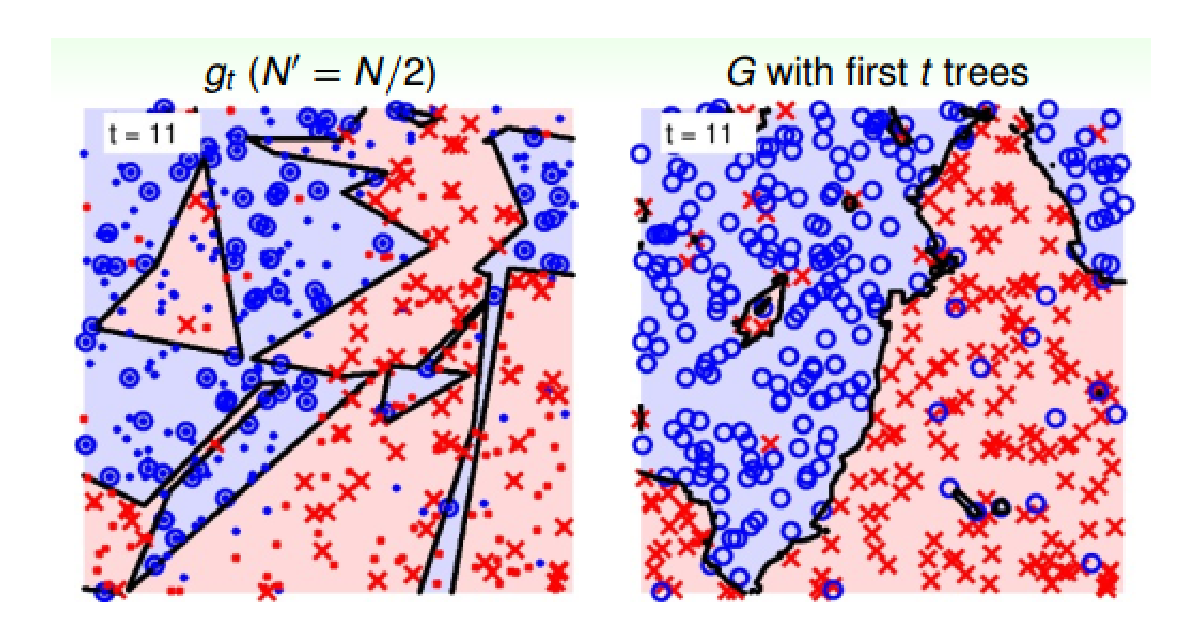

g_t ($N' = N/2$)
G with first t trees
t = 1
t = 1

当t=6时：

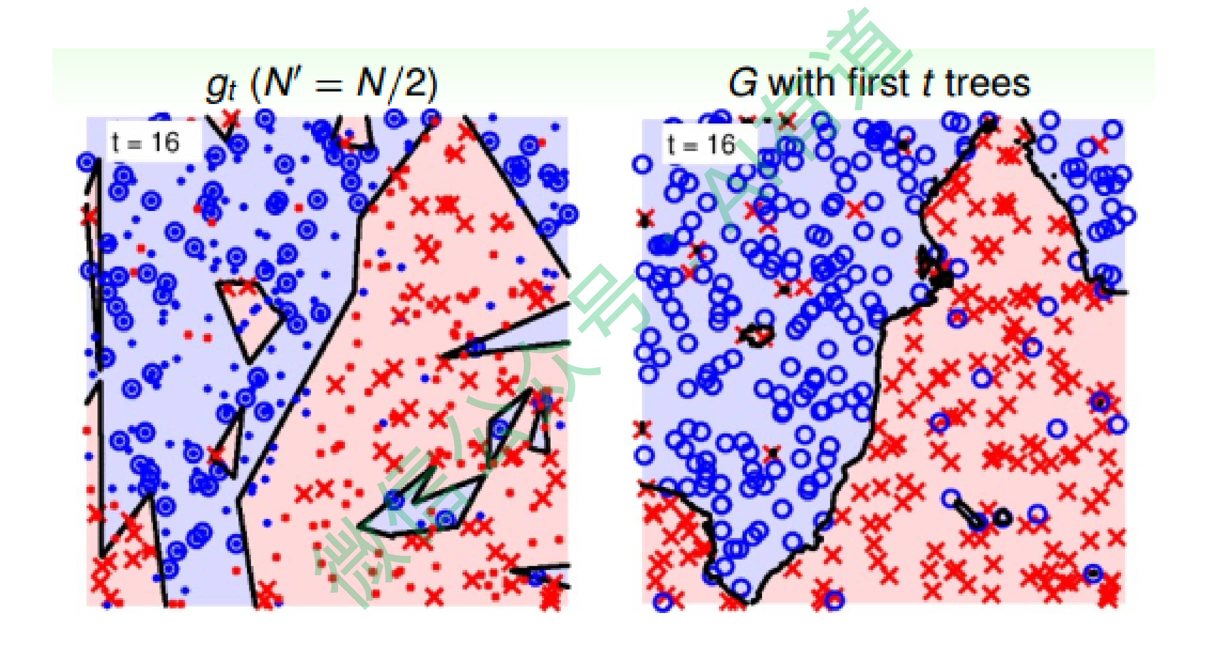

g_t ($N' = N/2$)
G with first t trees
t = 6
t = 6

当t=11时：



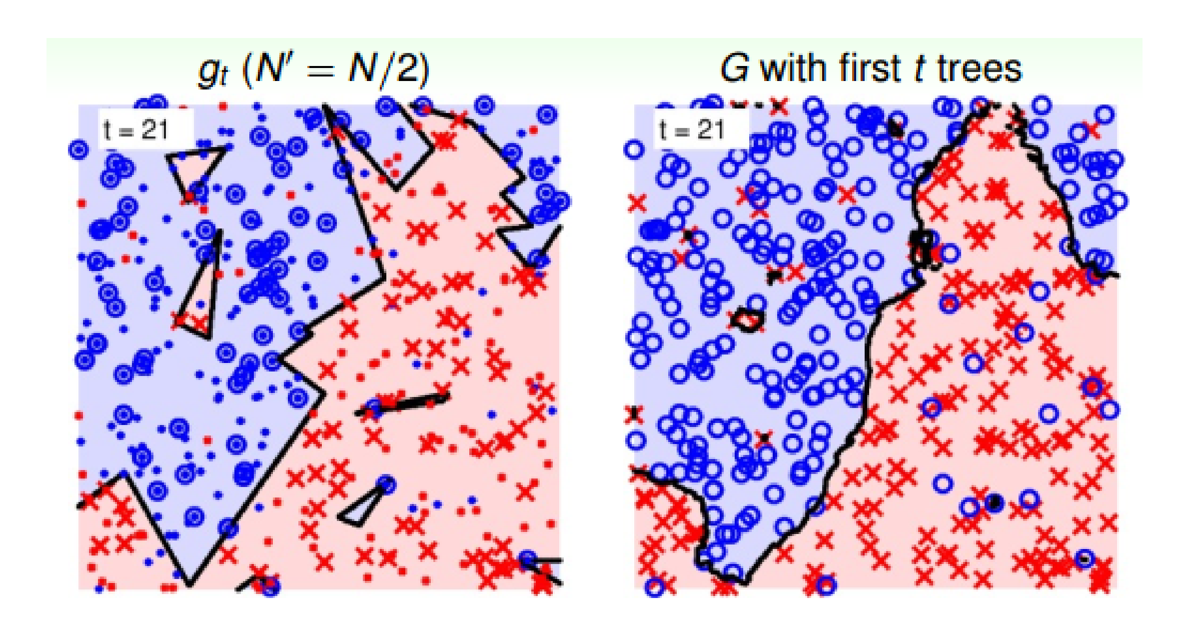

g_t $(N' = N/2)$
t = 11
G with first t trees
t = 11

当t=16时：


g_t $(N' = N/2)$
t = 16
G with first t trees
t = 16

当t=21时：





从上图中，我们发现21棵树的时候，随机noise的影响基本上能够修正和消除。这种bagging投票的机制能够保证较好的降噪性，从而得到比较稳定的结果。

经过以上三个例子，我们发现RF中，树的个数越多，模型越稳定越能表现得好。在实际应用中，应该尽可能选择更多的树。值得一提的是，RF的表现同时也与random seed有关，即随机的初始值也会影响RF的表现。

cons of RF: may need lots of trees **if the whole random process too unstable**
—should double-check **stability of G**
to ensure **enough trees**

总结：

本节课主要介绍了Random Forest算法模型。RF将bagging与decision tree结合起来，通过把众多的决策树组进行组合，构成森林的形式，利用投票机制让G表现最佳，分类模型更稳定。其中为了让decision tree的随机性更强一些，可以采用randomly projected subspaces操作，即将不同的features线性组合起来，从而进行各式各样的切割。同时，我们也介绍了可以使用OOB样本来进行self-validation，然后可以使用self-validation来对每个特征进行permutaion test，得到不同特征的重要性，从而进行feature selection。总的来说，RF算法能够得到比较平滑的边界，稳定性强，前提是有足够多的树。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程





林轩田《机器学习技法》课程笔记11 -- Gradient Boosted Decision Tree

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了Random Forest算法模型。Random Forest就是通过bagging的方式将许多不同的decision tree组合起来。除此之外，在decision tree中加入了各种随机性和多样性，比如不同特征的线性组合等。RF还可以使用OOB样本进行self-validation，而且可以通过permutation test进行feature selection。本节课将使用Adaptive Boosting的方法来研究decision tree的一些算法和模型。

Adaptive Boosted Decision Tree

Random Forest的算法流程我们上节课也详细介绍过，就是先通过bootstrapping“复制”原样本集D，得到新的样本集D'；然后对每个D'进行训练得到不同的decision tree和对应的g_t；最后再将所有的g_t通过uniform的形式组合起来，即以投票的方式得到G。这里采用的Bagging的方式，也就是把每个g_t的预测值直接相加。现在，如果将Bagging替换成AdaBoost，处理方式有些不同。首先每轮bootstrap得到的D'中每个样本会赋予不同的权重$u^{(t)}$；然后在每个decision tree中，利用这些权重训练得到最好的g_t；最后得出每个g_t所占的权重，线性组合得到G。这种模型称为AdaBoost-D Tree。

function RandomForest($\mathcal{D}$)
For $t = 1, 2, \ldots, T$
1. request size-N' data $\tilde{\mathcal{D}}_t$ by bootstrapping with $\mathcal{D}$
2. obtain tree g_t by Randomized-DTree($\tilde{\mathcal{D}}_t$)

return $G = \text{Uniform}(\{g_t\})$

function AdaBoost-DTree($\mathcal{D}$)
For $t = 1, 2, \ldots, T$
1. reweight data by $\mathbf{u}^{(t)}$
2. obtain tree g_t by DTree($\mathcal{D}, \mathbf{u}^{(t)}$)
3. calculate ‘vote’ α_t of g_t

return $G = \text{LinearHypo}(\{(g_t, \alpha_t)\})$

但是在AdaBoost-DTree中需要注意的一点是每个样本的权重$u^{(t)}$。我们知道，在Adaptive Boosting中进行了bootstrap操作，$u^{(t)}$表示D中每个样本在D'中出现的次数。但是在决策树模型中，例如C&RT算法中并没有引入$u^{(t)}$。那么，如何在决策树中



引入这些权重$u^{(t)}$来得到不同的g_t而又不改变原来的决策树算法呢?

在Adaptive Boosting中，我们使用了weighted algorithm，形如：

$$E_{in}^u(h)=\frac{1}{N}\sum_{n=1}^{N}u_n\cdot err(y_n,h(x_n))$$

每个犯错误的样本点乘以相应的权重，求和再平均，最终得到了$E_{in}^u(h)$。如果在决策树中使用这种方法，将当前分支下犯错误的点赋予权重，每层分支都这样做，会比较复杂，不易求解。为了简化运算，保持决策树算法本身的稳定性和封闭性，我们可以把决策树算法当成一个黑盒子，即不改变其结构，不对算法本身进行修改，而从数据来源D'上做一些处理。按照这种思想，我们来看权重u实际上表示该样本在bootstrap中出现的次数，反映了它出现的概率。那么可以根据u值，对原样本集D进行一次重新的随机sampling，也就是带权重的随机抽样。sampling之后，会得到一个新的D'，D'中每个样本出现的几率与它权重u所占的比例应该是差不多接近的。因此，使用带权重的sampling操作，得到了新的样本数据集D'，可以直接代入决策树进行训练，从而无需改变决策树算法结构。sampling可看成是bootstrap的反操作，这种对数据本身进行修改而不更改算法结构的方法非常重要!

'Weighted' Algorithm in Bagging	A General Randomized Base Algorithm
weights **u** expressed by bootstrap-sampled copies —request size-N' data $\tilde{\mathcal{D}}_t$ by bootstrapping with $\mathcal{D}$	weights **u** expressed by sampling proportional to u_n —request size-N' data $\tilde{\mathcal{D}}_t$ by sampling $\propto$ **u** on $\mathcal{D}$

所以，AdaBoost-DTree结合了AdaBoost和DTree，但是做了一点小小的改变，就是使用sampling替代权重$u^{(t)}$，效果是相同的。

AdaBoost-DTree: often via
AdaBoost + **sampling** $\propto \mathbf{u}^{(t)}$ + DTree($\tilde{\mathcal{D}}_t$)
without modifying DTree

上面我们通过使用sampling，将不同的样本集代入决策树中，得到不同的g_t。除此之外，我们还要确定每个g_t所占的权重α_t。之前我们在AdaBoost中已经介绍过，首先算出每个g_t的错误率ϵ_t，然后计算权重：



$$\alpha_t = ln \diamond_t = ln\sqrt{\frac{1-\epsilon_t}{\epsilon_t}}$$

如果现在有一棵完全长成的树（fully grown tree），由所有的样本x_n训练得到。若每个样本都不相同的话，一刀刀切割分支，直到所有的x_n都被完全分开。这时候，$E_{in}(g_t)=0$，加权的$E_{in}^u(g_t)=0$而且ϵ_t也为0，从而得到权重$\alpha_t=\infty$。$\alpha_t=\infty$表示该g_t所占的权重无限大，相当于它一个就决定了G结构，是一种autocracy，而其它的g_t对G没有影响。

if fully grown tree trained on all $\mathbf{x}_n$
$\Longrightarrow E_{in}(g_t)=0$ if all $\mathbf{x}_n$ different
$\Longrightarrow E_{in}^{\mathbf{u}}(g_t)=0$
$\Longrightarrow \epsilon_t=0$
$\Longrightarrow \alpha_t=\infty$ (**autocracy**!!)

显然$\alpha_t=\infty$不是我们想看到的，因为autocracy总是不好的，我们希望使用aggregation将不同的g_t结合起来，发挥集体智慧来得到最好的模型G。首先，我们来看一下什么原因造成了$\alpha_t=\infty$。有两个原因：一个是使用了所有的样本x_n进行训练；一个是树的分支过多，fully grown。针对这两个原因，我们可以对树做一些修剪（pruned），比如只使用一部分样本，这在sampling的操作中已经起到这类作用，因为必然有些样本没有被采样到。除此之外，我们还可以限制树的高度，让分支不要那么多，从而避免树fully grown。

need: **pruned** tree trained on **some** $\mathbf{x}_n$ to be **weak**

- **pruned**: usual pruning, or just **limiting tree height**
- **some**: **sampling** $\propto \mathbf{u}^{(t)}$

因此，AdaBoost-DTree使用的是pruned DTree，也就是说将这些预测效果较弱的树结合起来，得到最好的G，避免出现autocracy。

AdaBoost-DTree: often via AdaBoost + **sampling** $\propto \mathbf{u}^{(t)}$ + **pruned** DTree($\tilde{\mathcal{D}}$)

刚才我们说了可以限制树的高度，那索性将树的高度限制到最低，即只有1层高的时候，有什么特性呢？当树高为1的时候，整棵树只有两个分支，切割一次即可。如果



impurity是binary classification error的话，那么此时的AdaBoost-DTree就跟AdaBoost-Stump没什么两样。也就是说AdaBoost-Stump是AdaBoost-DTree的一种特殊情况。

DTree (C&RT) with height ≤ 1

learn branching criteria

$$b(\mathbf{x}) = \underset{\text{decision stumps } h(\mathbf{x})}{\operatorname{argmin}} \sum_{c=1}^{2} |\mathcal{D}_c \text{ with } h| \cdot \text{impurity}(\mathcal{D}_c \text{ with } h)$$

—if impurity = binary classification error,

just a decision stump, remember? :-)

值得一提是，如果树高为1时，通常较难遇到$\epsilon_t = 0$的情况，且一般不采用sampling的操作，而是直接将权重u代入到算法中。这是因为此时的AdaBoost-DTree就相当于是AdaBoost-Stump，而AdaBoost-Stump就是直接使用u来优化模型的。

Optimization View of AdaBoost

接下来，我们继续将继续探讨AdaBoost算法的一些奥妙之处。我们知道AdaBoost中的权重的迭代计算如下所示：

$$u_n^{(t+1)} = \begin{cases} u_n^{(t)} \cdot \blacklozenge_t & \textbf{if incorrect} \\ u_n^{(t)} / \blacklozenge_t & \textbf{if correct} \end{cases}$$

$$= u_n^{(t)} \cdot \blacklozenge_t^{-y_n g_t(\mathbf{x}_n)} = u_n^{(t)} \cdot \exp\left(-y_n \alpha_t g_t(\mathbf{x}_n)\right)$$

之前对于incorrect样本和correct样本，$u_n^{(t+1)}$的表达式不同。现在，把两种情况结合起来，将$u_n^{(t+1)}$写成一种简化的形式：

$$u_n^{(t+1)} = u_n^{(t)} \cdot \diamond_t^{-y_n g_t(x_n)} = u_n^{(t)} \cdot exp(-y_n \alpha_t g_t(x_n))$$

其中，对于incorrect样本，$y_n g_t(x_n) < 0$，对于correct样本，$y_n g_t(x_n) > 0$。从上式可以看出，$u_n^{(t+1)}$由$u_n^{(t)}$与某个常数相乘得到。所以，最后一轮更新的$u_n^{(T+1)}$可以写成$u_n^{(1)}$的级联形式，我们之前令$u_n^{(1)} = \frac{1}{N}$，则有如下推导：



$$u_n^{(T+1)}=u_n^{(1)}\cdot\prod_{t=1}^{T}exp(-y_n\alpha_tg_t(x_n))=\frac{1}{N}\cdot exp(-y_n\sum_{t=1}^{T}\alpha_tg_t(x_n))$$

上式中$\sum_{t=1}^{T}\alpha_tg_t(x_n)$被称为voting score，最终的模型$G=sign(\sum_{t=1}^{T}\alpha_tg_t(x_n))$。可以看出，在AdaBoost中，$u_n^{(T+1)}$与$exp(-y_n(voting\ score\ on\ x_n))$成正比。

$$u_n^{(T+1)}=u_n^{(1)}\cdot\prod_{t=1}^{T}\exp(-y_n\alpha_tg_t(\mathbf{x}_n))=\frac{1}{N}\cdot\exp\left(-y_n\sum_{t=1}^{T}\alpha_tg_t(\mathbf{x}_n)\right)$$

- recall: $G(\mathbf{x})=\text{sign}\left(\sum_{t=1}^{T}\alpha_tg_t(\mathbf{x})\right)$
- $\sum_{t=1}^{T}\alpha_tg_t(\mathbf{x})$: **voting score** of $\{g_t\}$ on $\mathbf{x}$

AdaBoost: $u_n^{(T+1)}\propto\exp(-y_n(\text{voting score on }\mathbf{x}_n))$

接下来我们继续看一下voting score中蕴含了哪些内容。如下图所示，voting score由许多$g_t(x_n)$乘以各自的系数α_t线性组合而成。从另外一个角度来看，我们可以把$g_t(x_n)$看成是对x_n的特征转换$\phi_i(x_n)$，α_t就是线性模型中的权重w_i。看到这里，我们回忆起之前SVM中，w与$\phi(x_n)$的乘积再除以w的长度就是margin，即点到边界的距离。另外，乘积项再与y_n相乘，表示点的位置是在正确的那一侧还是错误的那一侧。所以，回过头来，这里的voting score实际上可以看成是没有正规化（没有除以w的长度）的距离，即可以看成是该点到分类边界距离的一种衡量。从效果上说，距离越大越好，也就是说voting score要尽可能大一些。

linear blending = LinModel + hypotheses as transform + ~~constraints~~

$$G(\mathbf{x}_n)=\text{sign}\left(\overbrace{\sum_{t=1}^{T}\underbrace{\alpha_t}_{w_i}\underbrace{g_t(\mathbf{x}_n)}_{\phi_i(\mathbf{x}_n)}}^{\text{voting score}}\right)$$

and hard-margin SVM **margin** = $\frac{y_n\cdot(\mathbf{w}^T\phi(\mathbf{x}_n)+b)}{\|\mathbf{w}\|}$, **remember? :-)**



我们再来看，若voting score与y_n相乘，则表示一个有对错之分的距离。也就是说，如果二者相乘是负数，则表示该点在错误的一边，分类错误；如果二者相乘是正数，则表示该点在正确的一边，分类正确。所以，我们算法的目的就是让y_n与voting score的乘积是正的，而且越大越好。那么在刚刚推导的$u_n^{(T+1)}$中，得到$exp(-y_n(voting\ score))$越小越好，从而得到$u_n^{(T+1)}$越小越好。也就是说，如果voting score表现不错，与y_n的乘积越大的话，那么相应的$u_n^{(T+1)}$应该是最小的。

y_n(voting score) = signed & unnormalized **margin**

want y_n(voting score) **positive & large**

$\Leftrightarrow$ $\exp(-y_n(\text{voting score}))$ **small**

$\Leftrightarrow$ $u_n^{(T+1)}$ **small**

那么在AdaBoost中，随着每轮学习的进行，每个样本的$u_n^{(t)}$是逐渐减小的，直到$u_n^{(T+1)}$最小。以上是从单个样本点来看的。总体来看，所有样本的$u_n^{(T+1)}$之和应该也是最小的。我们的目标就是在最后一轮（T+1）学习后，让所有样本的$u_n^{(T+1)}$之和尽可能地小。$u_n^{(T+1)}$之和表示为如下形式：

claim: AdaBoost **decreases** $\sum_{n=1}^{N} u_n^{(t)}$ and thus somewhat **minimizes**

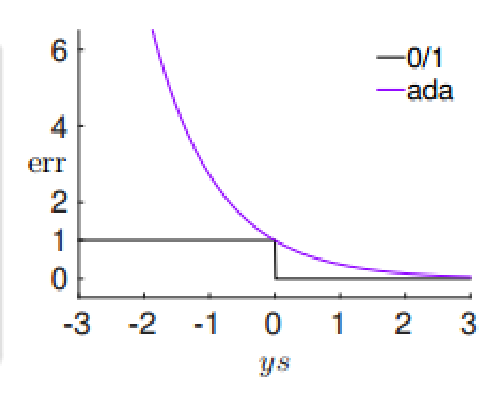
$$\sum_{n=1}^{N} u_n^{(T+1)} = \frac{1}{N} \sum_{n=1}^{N} \exp\left(-y_n \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)\right)$$

上式中，$\sum_{t=1}^{T} \alpha_t g_t(x_n)$被称为linear score，用s表示。对于0/1 error：若ys<0，则$err_{0/1}=1$；若ys>=0，则$err_{0/1}=0$。如下图右边黑色折线所示。对于上式中提到的指数error，即$e\hat{r}r_{ADA}(s,y)=exp(-ys)$，随着ys的增加，error单调下降，且始终落在0/1 error折线的上面。如下图右边蓝色曲线所示。很明显，$e\hat{r}r_{ADA}(s,y)$可以看成是0/1 error的上界。所以，我们可以使用$e\hat{r}r_{ADA}(s,y)$来替代0/1 error，能达到同样的效果。从这点来说，$\sum_{n=1}^{N} u_n^{(T+1)}$可以看成是一种error measure，而我们的目标就是让其最小化，求出最小值时对应的各个α_t和$g_t(x_n)$。



linear score $s = \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x}_n)$

- $\text{err}_{0/1}(s, y) = [\![ys \le 0]\!]$
- $\widehat{\text{err}}_{\text{ADA}}(s, y) = \exp(-ys)$:
  upper bound of $\text{err}_{0/1}$
  —called **exponential error measure**



$\widehat{\text{err}}_{\text{ADA}}$: **algorithmic error measure**
by **convex upper bound** of $\text{err}_{0/1}$

下面我们来研究如何让$\sum_{n=1}^{N} u_n^{(T+1)}$取得最小值，思考是否能用梯度下降（gradient descent）的方法来进行求解。我们之前介绍过gradient descent的核心是在某点处做一阶泰勒展开：

recall: gradient descent (remember? :-)), at iteration t

$$\min_{\|\mathbf{v}\|=1} E_{\text{in}}(\mathbf{w}_t + \eta \mathbf{v}) \approx \underbrace{E_{\text{in}}(\mathbf{w}_t)}_{\text{known}} + \underbrace{\eta}_{\text{given positive}} \mathbf{v}^T \underbrace{\nabla E_{\text{in}}(\mathbf{w}_t)}_{\text{known}}$$

其中，w_t是泰勒展开的位置，v是所要求的下降的最好方向，它是梯度$\nabla E_{in}(w_t)$的反方向，而η是每次前进的步长。则每次沿着当前梯度的反方向走一小步，就会不断逼近谷底（最小值）。这就是梯度下降算法所做的事情。

现在，我们对$\check{E}_{ADA}$做梯度下降算法处理，区别是这里的方向是一个函数g_t，而不是一个向量w_t。其实，函数和向量的唯一区别就是一个下标是连续的，另一个下标是离散的，二者在梯度下降算法应用上并没有大的区别。因此，按照梯度下降算法的展开式，做出如下推导：



at iteration t, to find g_t, solve

$$
\begin{aligned}
\min_{h} \quad \widehat{E}_{\text{ADA}} &= \frac{1}{N}\sum_{n=1}^{N} \exp\left(-y_n\left(\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n) + \eta h(\mathbf{x}_n)\right)\right) \\
&= \sum_{n=1}^{N} u_n^{(t)} \exp\left(-y_n \eta h(\mathbf{x}_n)\right) \\
&\overset{\text{taylor}}{\approx} \sum_{n=1}^{N} u_n^{(t)} \left(1 - y_n \eta h(\mathbf{x}_n)\right) = \sum_{n=1}^{N} u_n^{(t)} - \eta \sum_{n=1}^{N} u_n^{(t)} y_n h(\mathbf{x}_n)
\end{aligned}
$$

上式中，$h(x_n)$表示当前的方向，它是一个矩，η是沿着当前方向前进的步长。我们要求出这样的$h(x_n)$和η，使得$\check{E}_{ADA}$是在不断减小的。当$\check{E}_{ADA}$取得最小值的时候，那么所有的方向即最佳的$h(x_n)$和η就都解出来了。上述推导使用了在$-y_n\eta h(x_n)=0$处的一阶泰勒展开近似。这样经过推导之后，$\check{E}_{ADA}$被分解为两个部分，一个是前N个u之和$\sum_{n=1}^N u_n^{(t)}$，也就是当前所有的E_{in}之和；另外一个是包含下一步前进的方向$h(x_n)$和步进长度η的项$-\eta\sum_{n=1}^N u_n^{(t)} y_n h(x_n)$。$\check{E}_{ADA}$的这种形式与gradient descent的形式基本是一致的。

那么接下来，如果要最小化$\check{E}_{ADA}$的话，就要让第二项$-\eta\sum_{n=1}^N u_n^{(t)} y_n h(x_n)$越小越好。则我们的目标就是找到一个好的$h(x_n)$（即好的方向）来最小化$\sum_{n=1}^N u_n^{(t)}(-y_n h(x_n))$，此时先忽略步进长度$\eta$。

finding good h (function direction) $\Leftrightarrow$ minimize $\sum_{n=1}^{N} u_n^{(t)} \left(-y_n h(\mathbf{x}_n)\right)$

对于binary classification，y_n和$h(x_n)$均限定取值-1或+1两种。我们对$\sum_{n=1}^N u_n^{(t)}(-y_n h(x_n))$做一些推导和平移运算：



for binary classification, where y_n and $h(\mathbf{x}_n)$ both $\in \{-1, +1\}$:

$$\begin{aligned}\sum_{n=1}^{N} u_n^{(t)} (-y_n h(\mathbf{x}_n)) &= \sum_{n=1}^{N} u_n^{(t)} \begin{cases} -1 & \text{if } y_n = h(\mathbf{x}_n) \\ +1 & \text{if } y_n \neq h(\mathbf{x}_n) \end{cases} \\ &= -\sum_{n=1}^{N} u_n^{(t)} + \sum_{n=1}^{N} u_n^{(t)} \begin{cases} 0 & \text{if } y_n = h(\mathbf{x}_n) \\ 2 & \text{if } y_n \neq h(\mathbf{x}_n) \end{cases} \\ &= -\sum_{n=1}^{N} u_n^{(t)} + 2E_{\text{in}}^{\mathbf{u}^{(t)}}(h) \cdot N \end{aligned}$$

—who minimizes $E_{\text{in}}^{\mathbf{u}^{(t)}}(h)$? $\mathcal{A}$ **in AdaBoost! :-)**

最终$\sum_{n=1}^{N} u_n^{(t)}(-y_n h(x_n))$化简为两项组成，一项是$-\sum_{n=1}^{N} u_n^{(t)}$；另一项是$2E_{in}^{u(t)}(h) \cdot N$。则最小化$\sum_{n=1}^{N} u_n^{(t)}(-y_n h(x_n))$就转化为最小化$E_{in}^{u(t)}(h)$。要让$E_{in}^{u(t)}(h)$最小化，正是由AdaBoost中的base algorithm所做的事情。所以说，AdaBoost中的base algorithm正好帮我们找到了梯度下降中下一步最好的函数方向。

$\mathcal{A}$: **good** $g_t = h$ for 'gradient descent'

以上就是从数学上，从gradient descent角度验证了AdaBoost中使用base algorithm得到的g_t就是让$\check{E}_{ADA}$减小的方向，只不过这个方向是一个函数而不是向量。

在解决了方向问题后，我们需要考虑步进长度η如何选取。方法是在确定方向g_t后，选取合适的η，使$\check{E}_{ADA}$取得最小值。也就是说，把$\check{E}_{ADA}$看成是步进长度η的函数，目标是找到$\check{E}_{ADA}$最小化时对应的η值。

AdaBoost finds g_t by approximately $\min_{h} \widehat{E}_{\text{ADA}} = \sum_{n=1}^{N} u_n^{(t)} \exp(-y_n \eta h(\mathbf{x}_n))$

after finding g_t, how about $\min_{\eta} \widehat{E}_{\text{ADA}} = \sum_{n=1}^{N} u_n^{(t)} \exp(-y_n \eta g_t(\mathbf{x}_n))$

目的是找到在最佳方向上的最大步进长度，也就是steepest decent。我们先把$\check{E}_{ADA}$表达式写下来：

$$\check{E}_{ADA} = \sum_{n=1}^{N} u_n^{(t)} exp(-y_n \eta g_t(x_n))$$



上式中，有两种情况需要考虑：

- $y_n = g_t(x_n)$：$u_n^{(t)} exp(-\eta)$ **correct**
- $y_n \neq g_t(x_n)$：$u_n^{(t)} exp(+\eta)$ **incorrect**

经过推导，可得：

$$\check{E}_{ADA} = (\sum_{n=1}^{N} u_n^{(t)}) \cdot ((1-\epsilon_t)exp(-\eta) + \epsilon_t\ exp(+\eta))$$

- optimal η_t somewhat '**greedily faster**' than fixed (small) η —called **steepest** descent for optimization
- two cases inside summation:
  - $y_n = g_t(\mathbf{x}_n)$: $u_n^{(t)} \exp(-\eta)$ (correct)
  - $y_n \neq g_t(\mathbf{x}_n)$: $u_n^{(t)} \exp(+\eta)$ (incorrect)
- $\widehat{E}_{\mathrm{ADA}} = \left(\sum_{n=1}^{N} u_n^{(t)}\right) \cdot \left((1-\epsilon_t) \exp(-\eta) + \epsilon_t \exp(+\eta)\right)$

然后对η求导，令$\frac{\partial \check{E}_{ADA}}{\partial \eta} = 0$，得：

$$\eta_t = ln\sqrt{\frac{1-\epsilon_t}{\epsilon_t}} = \alpha_t$$

由此看出，最大的步进长度就是α_t，即AdaBoost中计算g_t所占的权重。所以，AdaBoost算法所做的其实是在gradient descent上找到下降最快的方向和最大的步进长度。这里的方向就是g_t，它是一个函数，而步进长度就是α_t。也就是说，在AdaBoost中确定g_t和α_t的过程就相当于在gradient descent上寻找最快的下降方向和最大的步进长度。

Gradient Boosting

前面我们从gradient descent的角度来重新介绍了AdaBoost的最优化求解方法。整个过程可以概括为：



AdaBoost

$$\min_{\eta} \min_{h} \frac{1}{N} \sum_{n=1}^{N} \exp\left(-y_n \left(\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n) + \eta h(\mathbf{x}_n)\right)\right)$$

with binary-output hypothesis h

以上是针对binary classification问题。如果往更一般的情况进行推广，对于不同的error function，比如logistic error function或者regression中的squared error function，那么这种做法是否仍然有效呢？这种情况下的GradientBoost可以写成如下形式：

GradientBoost

$$\min_{\eta} \min_{h} \frac{1}{N} \sum_{n=1}^{N} \text{err}\left(\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n) + \eta h(\mathbf{x}_n), y_n\right)$$

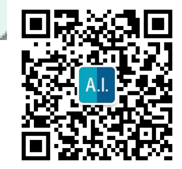
with any hypothesis h (usually real-output hypothesis)

仍然按照gradient descent的思想，上式中，$h(x_n)$是下一步前进的方向，η是步进长度。此时的error function不是前面所讲的exp了，而是任意的一种error function。因此，对应的hypothesis也不再是binary classification，最常用的是实数输出的hypothesis，例如regression。最终的目标也是求解最佳的前进方向$h(x_n)$和最快的步进长度η。

GradientBoost: allows **extension to different** err for regression/soft classification/etc.

接下来，我们就来看看如何求解regression的GradientBoost问题。它的表达式如下所示：

$$\min_{\eta} \min_{h} \frac{1}{N} \sum_{n=1}^{N} \text{err}\Big(\underbrace{\sum_{\tau=1}^{t-1} \alpha_\tau g_\tau(\mathbf{x}_n)}_{s_n} + \eta h(\mathbf{x}_n), y_n\Big) \quad \text{with } \text{err}(s, y) = (s - y)^2$$



利用梯度下降的思想，我们把上式进行一阶泰勒展开，写成梯度的形式：

$$\min_h \ldots \overset{\text{taylor}}{\approx} \min_h \frac{1}{N}\sum_{n=1}^{N}\underbrace{\text{err}(s_n, y_n)}_{\text{constant}} + \frac{1}{N}\sum_{n=1}^{N}\eta h(\mathbf{x}_n)\left.\frac{\partial \text{err}(s, y_n)}{\partial s}\right|_{s=s_n}$$

$$= \min_h \text{constants} + \frac{\eta}{N}\sum_{n=1}^{N} h(\mathbf{x}_n)\cdot 2(s_n - y_n)$$

上式中，由于regression的error function是squared的，所以，对s的导数就是$2(s_n - y_n)$。其中标注灰色的部分表示常数，对最小化求解并没有影响，所以可以忽略。很明显，要使上式最小化，只要令$h(x_n)$是梯度$2(s_n - y_n)$的反方向就行了，即$h(x_n) = -2(s_n - y_n)$。但是直接这样赋值，并没有对$h(x_n)$的大小进行限制，一般不直接利用这个关系求出$h(x_n)$。

$$\min_h \text{constants} + \frac{\eta}{N}\sum_{n=1}^{N} 2h(\mathbf{x}_n)(s_n - y_n)$$

实际上$h(x_n)$的大小并不重要，因为有步进长度η。那么，我们上面的最小化问题中需要对$h(x_n)$的大小做些限制。限制$h(x_n)$的一种简单做法是把$h(x_n)$的大小当成一个惩罚项（$h^2(x_n)$）添加到上面的最小化问题中，这种做法与regularization类似。如下图所示，经过推导和整理，忽略常数项，我们得到最关心的式子是：

$$min \sum_{n=1}^{N}((h(x_n) - (y_n - s_n))^2)$$

上式是一个完全平方项之和，$y_n - s_n$表示当前第n个样本真实值和预测值的差，称之为余数。余数表示当前预测能够做到的效果与真实值的差值是多少。那么，如果我们想要让上式最小化，求出对应的$h(x_n)$的话，只要让$h(x_n)$尽可能地接近余数$y_n - s_n$即可。在平方误差上尽可能接近其实很简单，就是使用regression的方法，对所有N个点$(x_n, y_n - s_n)$做squared-error的regression，得到的回归方程就是我们要求的$g_t(x_n)$。



- magnitude of h does not matter: because η will be optimized next
- penalize large magnitude to avoid naïve solution

$$\min_{h} \quad \text{constants} + \frac{\eta}{N}\sum_{n=1}^{N}\left(2h(\mathbf{x}_n)(s_n - y_n) + (h(\mathbf{x}_n))^2\right)$$

$$= \quad \text{constants} + \frac{\eta}{N}\sum_{n=1}^{N}\left(\text{constant} + (h(\mathbf{x}_n) - (y_n - s_n))^2\right)$$

- solution of penalized approximate functional gradient: squared-error regression on $\{(\mathbf{x}_n, \underbrace{y_n - s_n}_{\text{residual}})\}$

以上就是使用GradientBoost的思想来解决regression问题的方法，其中应用了一个非常重要的概念，就是余数$y_n - s_n$。根据这些余数做regression，得到好的矩$g_t(x_n)$，方向函数$g_t(x_n)$也就是由余数决定的。

> GradientBoost for regression:
> find $g_t = h$ by regression with residuals

在求出最好的方向函数$g_t(x_n)$之后，就要来求相应的步进长度η。表达式如下：

after finding $g_t = h$,

$$\min_{\eta} \frac{1}{N}\sum_{n=1}^{N}\text{err}\Big(\underbrace{\sum_{\tau=1}^{t-1}\alpha_\tau g_\tau(\mathbf{x}_n)}_{s_n} + \eta g_t(\mathbf{x}_n), y_n\Big) \text{ with } \text{err}(s, y) = (s - y)^2$$

同样，对上式进行推导和化简，得到如下表达式：

$$\min_{\eta} \quad \frac{1}{N}\sum_{n=1}^{N}(s_n + \eta g_t(\mathbf{x}_n) - y_n)^2 = \frac{1}{N}\sum_{n=1}^{N}((y_n - s_n) - \eta g_t(\mathbf{x}_n))^2$$

—one-variable linear regression on $\{(g_t\text{-transformed input, residual})\}$

上式中也包含了余数$y_n - s_n$，其中$g_t(x_n)$可以看成是x_n的特征转换，是已知量。那



么，如果我们想要让上式最小化，求出对应的η的话，只要让$\eta g_t(x_n)$尽可能地接近余数$y_n - s_n$即可。显然，这也是一个regression问题，而且是一个很简单的形如y=ax的线性回归，只有一个未知数η。只要对所有N个点$(\eta g_t(x_n), y_n - s_n)$做squared-error的linear regression，利用梯度下降算法就能得到最佳的η。

将上述这些概念合并到一起，我们就得到了一个最终的演算法Gradient Boosted Decision Tree(GBDT)。可能有人会问，我们刚才一直没有说到Decison Tree，只是讲到了GradientBoost啊？下面我们来看看Decison Tree究竟是在哪出现并使用的。其实刚刚我们在计算方向函数g_t的时候，是对所有N个点$(x_n, y_n - s_n)$做squared-error的regression。那么这个回归算法就可以是决策树C&RT模型（决策树也可以用来做regression）。这样，就引入了Decision Tree，并将GradientBoost和Decision Tree结合起来，构成了真正的GBDT算法。GBDT算法的基本流程图如下所示：

Gradient Boosted Decision Tree (GBDT)

$s_1 = s_2 = \ldots = s_N = 0$
for $t = 1, 2, \ldots, T$
1. obtain g_t by $\mathcal{A}(\{(\mathbf{x}_n, y_n - s_n)\})$ where $\mathcal{A}$ is a (squared-error) regression algorithm
   —**how about sampled and pruned C&RT?**
2. compute $\alpha_t = \text{OneVarLinearRegression}(\{(g_t(\mathbf{x}_n), y_n - s_n)\})$
3. update $s_n \leftarrow s_n + \alpha_t g_t(\mathbf{x}_n)$

return $G(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t g_t(\mathbf{x})$

值得注意的是，s_n的初始值一般均设为0，即$s_1 = s_2 = \cdots = s_N = 0$。每轮迭代中，方向函数$g_t$通过C&RT算法做regression，进行求解；步进长度η通过简单的单参数线性回归进行求解；然后每轮更新s_n的值，即$s_n \leftarrow s_n + \alpha_t g_t(x_n)$。T轮迭代结束后，最终得到$G(x) = \sum_{t=1}^{T} \alpha_t g_t(x)$。

值得一提的是，本节课第一部分介绍的AdaBoost-DTree是解决binary classification问题，而此处介绍的GBDT是解决regression问题。二者具有一定的相似性，可以说GBDT就是AdaBoost-DTree的regression版本。

GBDT: 'regression sibling' of AdaBoost-DTree
—**popular in practice**

Summary ofAggregation Models



从机器学习技法课程的第7节课笔记到现在的第11节课笔记，我们已经介绍完所有的aggregation模型了。接下来，我们将对这些内容进行一个简单的总结和概括。

首先，我们介绍了blending。blending就是将所有已知的g_t aggregate结合起来，发挥集体的智慧得到G。值得注意的一点是这里的g_t都是已知的。blending通常有三种形式：

- **uniform：简单地计算所有g_t的平均值**
- **non-uniform：所有g_t的线性组合**
- **conditional：所有g_t的非线性组合**

其中，uniform采用投票、求平均的形式更注重稳定性；而non-uniform和conditional追求的更复杂准确的模型，但存在过拟合的危险。

blending: aggregate after getting **diverse** g_t

uniform	non-uniform	conditional
simple voting/averaging of g_t	linear model on g_t-transformed inputs	nonlinear model on g_t-transformed inputs

uniform for 'stability';
non-uniform/conditional **carefully** for 'complexity'

刚才讲的blending是建立在所有g_t已知的情况。那如果所有g_t未知的情况，对应的就是learning模型，做法就是一边学g_t，一边将它们结合起来。learning通常也有三种形式（与blending的三种形式一一对应）：

- **Bagging：通过bootstrap方法，得到不同g_t，计算所有g_t的平均值**
- **AdaBoost：通过bootstrap方法，得到不同g_t，所有g_t的线性组合**
- **Decision Tree：通过数据分割的形式得到不同的g_t，所有g_t的非线性组合**

然后，本节课我们将AdaBoost延伸到另一个模型GradientBoost。对于regression问题，GradientBoost通过residual fitting的方式得到最佳的方向函数g_t和步进长度η。



learning: aggregate as well as getting **diverse** g_t

Bagging	AdaBoost	Decision Tree
diverse g_t by bootstrapping; uniform vote by nothing :-)	diverse g_t by reweighting; linear vote by steepest search	diverse g_t by data splitting; conditional vote by branching

GradientBoost
diverse g_t by residual fitting; linear vote by steepest search

boosting-like algorithms most popular

除了这些基本的aggregation模型之外，我们还可以把某些模型结合起来得到新的aggregation模型。例如，Bagging与Decision Tree结合起来组成了Random Forest。Random Forest中的Decision Tree是比较“茂盛”的树，即每个树的g_t都比较强一些。AdaBoost与Decision Tree结合组成了AdaBoost-DTree。AdaBoost-DTree的Decision Tree是比较“矮弱”的树，即每个树的g_t都比较弱一些，由AdaBoost将所有弱弱的树结合起来，让综合能力更强。同样，GradientBoost与Decision Tree结合就构成了经典的算法GBDT。

Bagging	AdaBoost	Decision Tree

Random Forest	AdaBoost-DTree
randomized bagging + ‘strong’ DTree	AdaBoost + ‘weak’ DTree

GradientBoost

GBDT
GradientBoost + ‘weak’ DTree

all three frequently used in practice

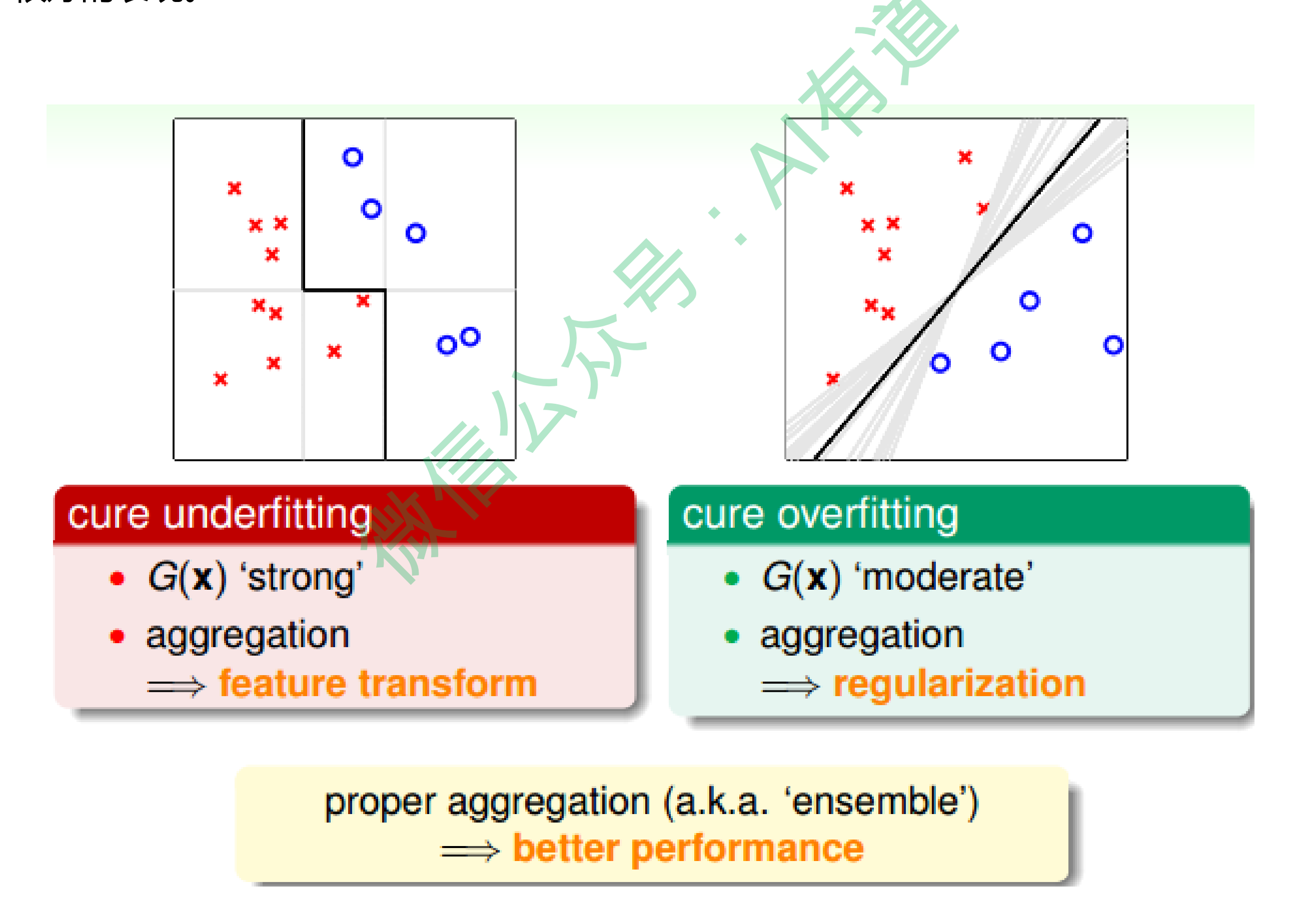


Aggregation的核心是将所有的g_t结合起来，融合到一起，即集体智慧的思想。这种做法之所以能得到很好的模型G，是因为aggregation具有两个方面的优点：cure underfitting和cure overfitting。

第一，aggregation models有助于防止欠拟合（underfitting）。它把所有比较弱的g_t结合起来，利用集体智慧来获得比较好的模型G。aggregation就相当于是feature transform，来获得复杂的学习模型。

第二，aggregation models有助于防止过拟合（overfitting）。它把所有g_t进行组合，容易得到一个比较中庸的模型，类似于SVM的large margin一样的效果，从而避免一些极端情况包括过拟合的发生。从这个角度来说，aggregation起到了regularization的效果。

由于aggregation具有这两个方面的优点，所以在实际应用中aggregation models都有很好的表现。



总结

本节课主要介绍了Gradient Boosted Decision Tree。首先讲如何将AdaBoost与Decision Tree结合起来，即通过sampling和pruning的方法得到AdaBoost-D Tree模型。然后，我们从optimization的角度来看AdaBoost，找到好的hypothesis也就是找到一个好的方向，找到权重α也就是找到合适的步进长度。接着，我们从binary

classification的0/1 error推广到其它的error function，从Gradient Boosting角度推导了regression的squared error形式。Gradient Boosting其实就是不断迭代，做residual fitting。并将其与Decision Tree算法结合，得到了经典的GBDT算法。最后，我们将所有的aggregation models做了总结和概括，这些模型有的能防止欠拟合有的能防止过拟合，应用十分广泛。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程

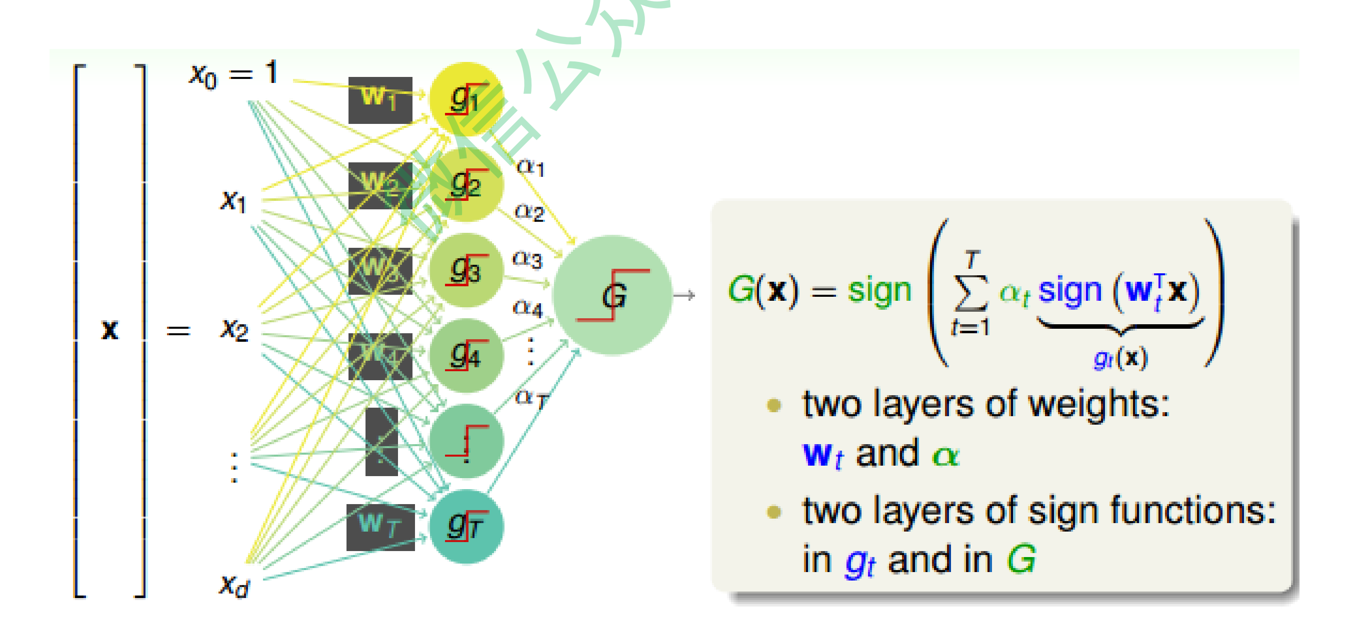
林轩田《机器学习技法》课程笔记12 -- Neural Network

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了Gradient Boosted Decision Tree。GBDT通过使用functional gradient的方法得到一棵一棵不同的树，然后再使用steepest descent的方式给予每棵树不同的权重，最后可以用来处理任何而定error measure。上节课介绍的GBDT是以regression为例进行介绍的，使用的是squared error measure。本节课讲介绍一种出现时间较早，但当下又非常火的一种机器算法模型，就是神经网络（Neural Network）。

Motivation

在之前的机器学习基石课程中，我们就接触过Perceptron模型了，例如PLA算法。Perceptron就是在矩$g_t(x)$外面加上一个sign函数，取值为{-1,+1}。现在，如果把许多perceptrons线性组合起来，得到的模型G就如下图所示：



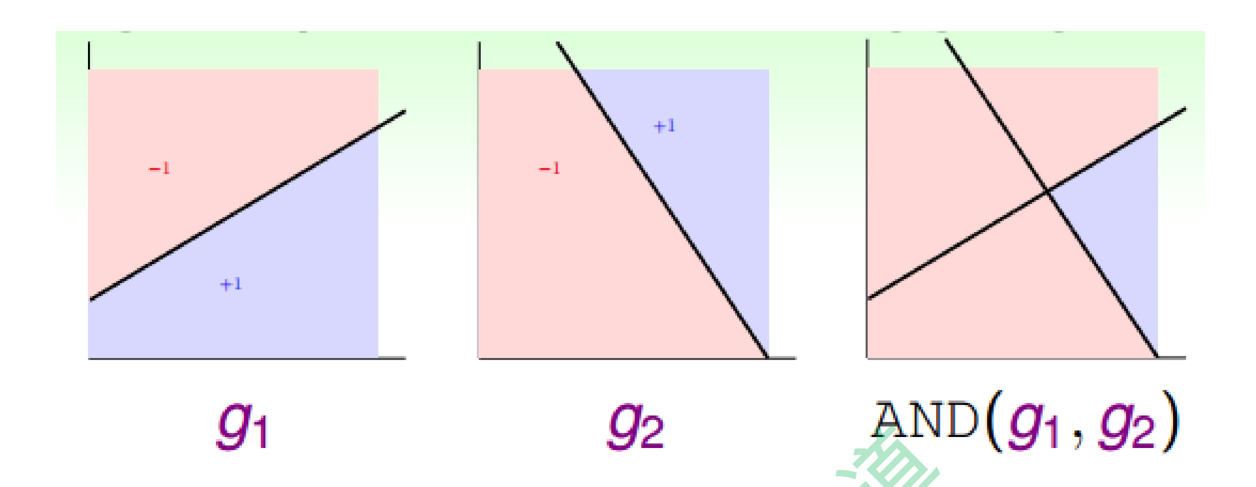
将左边的输入$(x_0, x_1, x_2, \cdots, x_d)$与T个不同的权重$(w_1, w_2, \cdots, w_T)$相乘（每个$w_i$是d+1维的），得到T个不同的perceptrons为$(g_1, g_2, \cdots, g_T)$。最后，每个g_t给予不同的权重$(\alpha_1, \alpha_2, \cdots, \alpha_T)$，线性组合得到G。G也是一个perceptron模型。

从结构上来说，上面这个模型包含了两层的权重，分别是w_t和α。同时也包含了两层的sign函数，分别是g_t和G。那么这样一个由许多感知机linear aggregation的模型能



实现什么样的boundary呢?

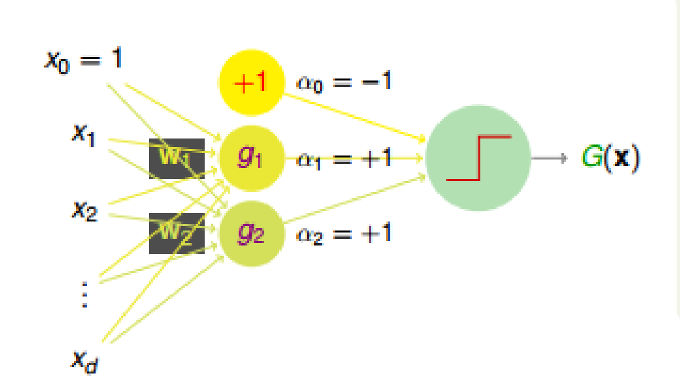
举个简单的例子，如下图所示，g_1和g_2分别是平面上两个perceptrons。其中，红色表示-1，蓝色表示+1。这两个perceptrons线性组合可能得到下图右侧的模型，这表示的是g_1和g_2进行与（AND）的操作，蓝色区域表示+1。



如何通过感知机模型来实现上述的$AND(g_1, g_2)$逻辑操作呢？一种方法是令第二层中的$\alpha_0 = -1, \alpha_1 = +1, \alpha_2 = +1$。这样，G(x)就可表示为：

$$G(x) = sign(-1 + g_1(x) + g_2(x))$$

g_1和g_2的取值是{-1,+1}，当$g_1 = -1$，$g_2 = -1$时，G(x)=0；当$g_1 = -1$，$g_2 = +1$时，G(x)=0；当$g_1 = +1$，$g_2 = -1$时，G(x)=0；当$g_1 = +1$，$g_2 = +1$时，G(x)=1。感知机模型如下所示：



$G(\mathbf{x}) = \text{sign}(-1 + g_1(\mathbf{x}) + g_2(\mathbf{x}))$

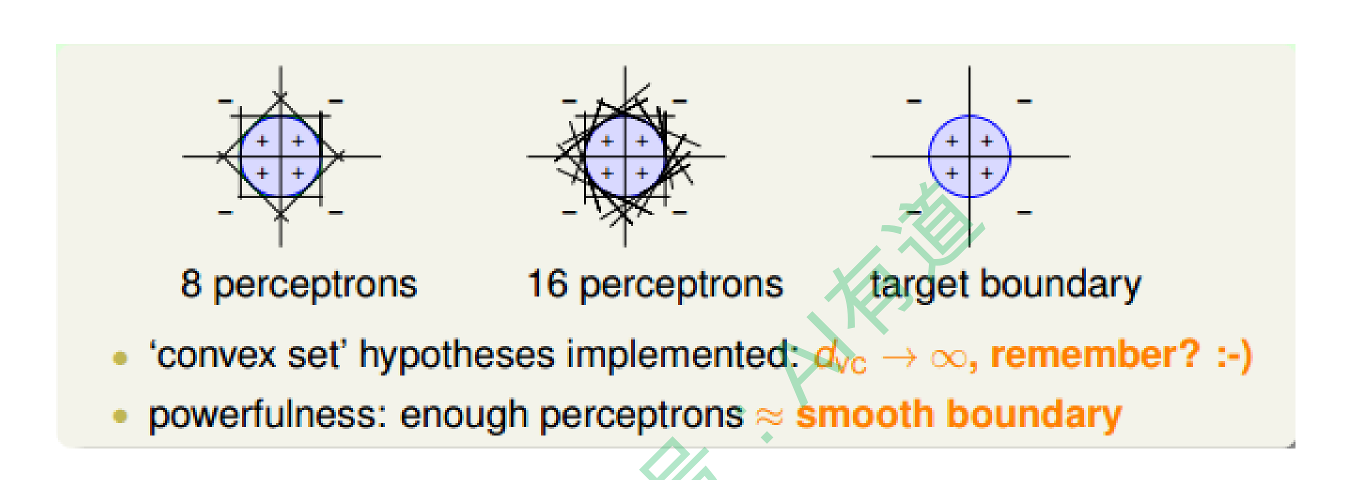
- $g_1(\mathbf{x}) = g_2(\mathbf{x}) = +1$ (TRUE): $G(\mathbf{x}) = +1$ (TRUE)
- otherwise: $G(\mathbf{x}) = -1$ (FALSE)
- $G \equiv \text{AND}(g_1, g_2)$

这个例子说明了一些简单的线性边界，如上面的g_1和g_2，在经过一层感知机模型，经线性组合后，可以得到一些非线性的复杂边界（AND运算）G(x)。

除此之外，或（OR）运算和非（NOT）运算都可以由感知机建立相应的模型，非常简单。

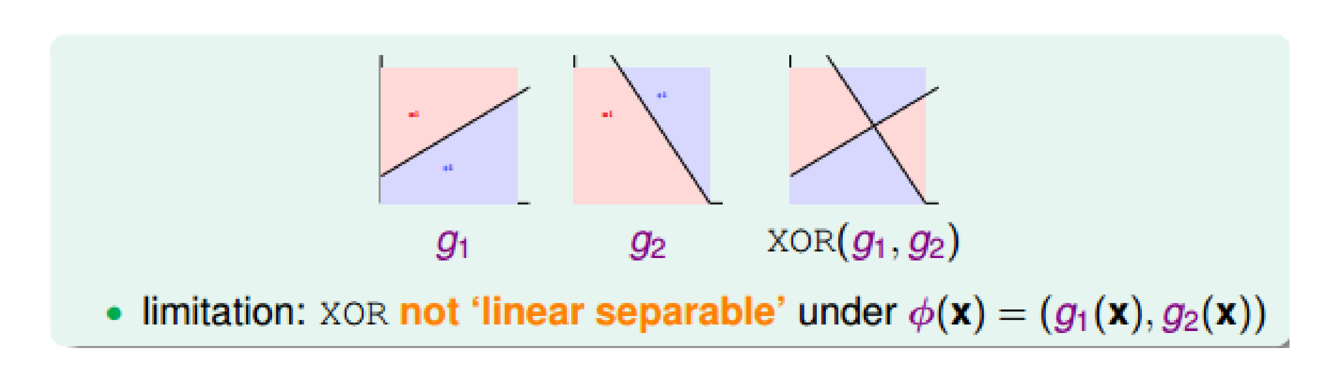


所以说，linear aggregation of perceptrons实际上是非常powerful的模型同时也是非常complicated模型。再看下面一个例子，如果二维平面上有个圆形区域，圆内表示+1，圆外表示-1。这样复杂的圆形边界是没有办法使用单一perceptron来解决的。如果使用8个perceptrons，用刚才的方法线性组合起来，能够得到一个很接近圆形的边界（八边形）。如果使用16个perceptrons，那么得到的边界更接近圆形（十六边形）。因此，使用的perceptrons越多，就能得到各种任意的convex set，即凸多边形边界。之前我们在机器学习基石中介绍过，convex set的VC Dimension趋向于无穷大（2^N）。这表示只要perceptrons够多，我们能得到任意可能的情况，可能的模型。但是，这样的坏处是模型复杂度可能会变得很大，从而造成过拟合（overfitting）。



总的来说，足够数目的perceptrons线性组合能够得到比较平滑的边界和稳定的模型，这也是aggregation的特点之一。

但是，也有单层perceptrons线性组合做不到的事情。例如刚才我们将的AND、OR、NOT三种逻辑运算都可以由单层perceptrons做到，而如果是异或（XOR）操作，就没有办法只用单层perceptrons实现。这是因为XOR得到的是非线性可分的区域，如下图所示，没有办法由g_1和g_2线性组合实现。所以说linear aggregation of perceptrons模型的复杂度还是有限制的。



那么，为了实现XOR操作，可以使用多层perceptrons，也就是说一次transform不行，我们就用多层的transform，这其实就是Basic Neural Network的基本原型。下面

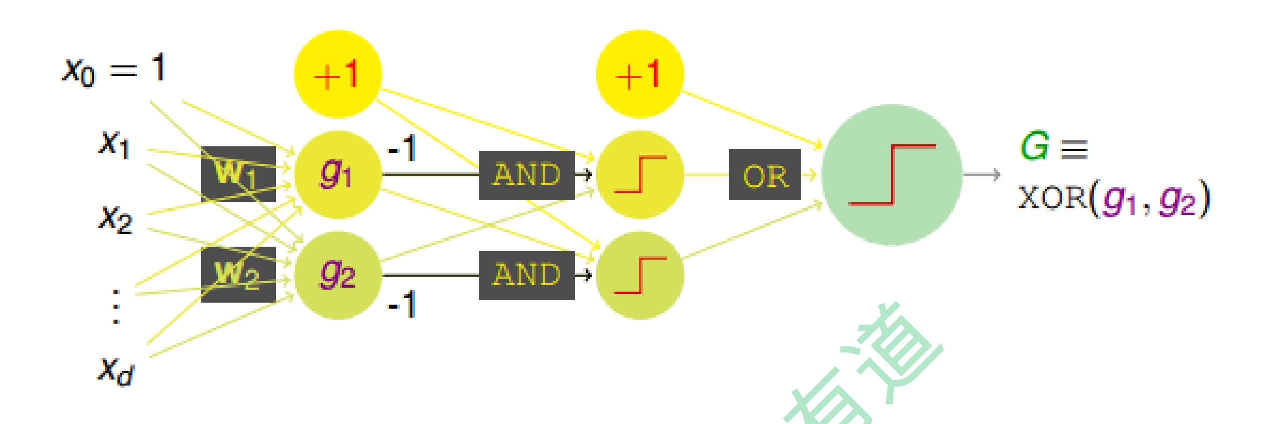


我们就尝试使用两层perceptrons来实现XOR的操作。

首先，根据布尔运算，异或XOR操作可以拆分成：

$$XOR(g_1, g_2) = OR(AND(-g_1, g_2), AND(g_1, -g_2))$$

这种拆分实际上就包含了两层transform。第一层仅有AND操作，第二层是OR操作。这种两层的感知机模型如下所示：



这样，从AND操作到XOR操作，从简单的aggregation of perceptrons到multi-layer perceptrons，感知机层数在增加，模型的复杂度也在增加，使最后得到的G能更容易解决一些非线性的复杂问题。这就是基本神经网络的基本模型。

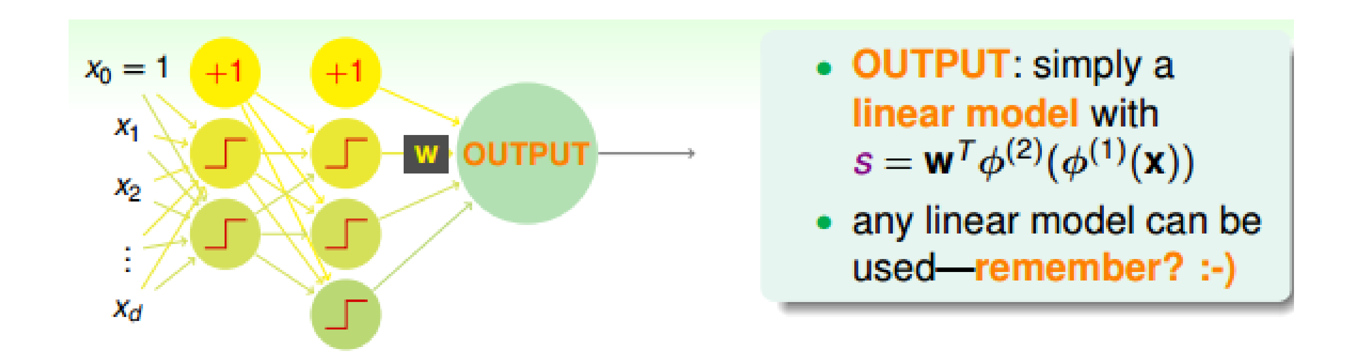
perceptron (simple)
⟹ aggregation of perceptrons (powerful)
⟹ **multi-layer perceptrons (more powerful)**

顺便提一下，这里所说的感知机模型实际上就是在模仿人类的神经元模型（这就是Neural Network名称的由来）。感知机模型每个节点的输入就对应神经元的树突dendrite，感知机每个节点的输出就对应神经元的轴突axon。

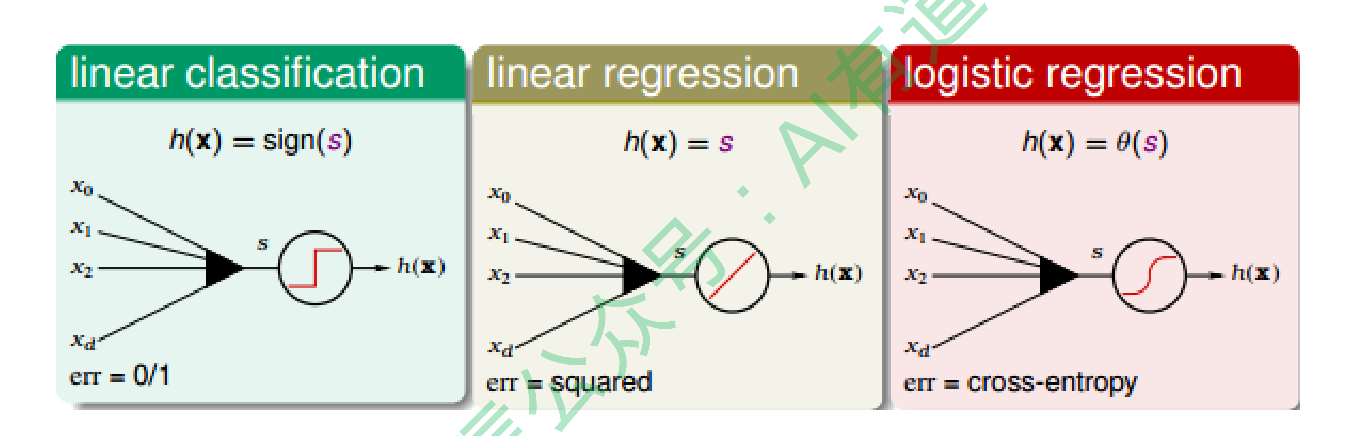
Neural Network Hypothesis

上一部分我们介绍的这种感知机模型其实就是Neural Network。输入部分经过一层一层的运算，相当于一层一层的transform，最后通过最后一层的权重，得到一个分数score。即在OUTPUT层，输出的就是一个线性模型。得到s后，下一步再进行处理。





我们之前已经介绍过三种线性模型：linear classification，linear regression，logistic regression。那么，对于OUTPUT层的分数s，根据具体问题，可以选择最合适的线性模型。如果是binary classification问题，可以选择linear classification模型；如果是linear regression问题，可以选择linear regression模型；如果是soft classification问题，则可以选择logistic regression模型。本节课接下来将以linear regression为例，选择squared error来进行衡量。



上面讲的是OUTPUT层，对于中间层，每个节点对应一个perceptron，都有一个transform运算。上文我们已经介绍过的transformation function是阶梯函数sign()。那除了sign()函数外，有没有其他的transformation function呢？

如果每个节点的transformation function都是线性运算（跟OUTPUT端一样），那么由每个节点的线性模型组合成的神经网络模型也必然是线性的。这跟直接使用一个线性模型在效果上并没有什么差异，模型能力不强，反而花费了更多不必要的力气。所以一般来说，中间节点不会选择线性模型。

如果每个节点的transformation function都是阶梯函数（即sign()函数）。这是一个非线性模型，但是由于阶梯函数是离散的，并不是处处可导，所以在优化计算时比较难处理。所以，一般也不选择阶梯函数作为transformation function。

既然线性函数和阶梯函数都不太适合作为transformation function，那么最常用的一种transformation function就是tanh(s)，其表达式如下：



$$tanh(s)=\frac{exp(s)-exp(-s)}{exp(s)+exp(-s)}$$

tanh(s)函数是一个平滑函数，类似“s”型。当|s|比较大的时候，tanh(s)与阶梯函数相近；当|s|比较小的时候，tanh(s)与线性函数比较接近。从数学上来说，由于处处连续可导，便于最优化计算。而且形状上类似阶梯函数，具有非线性的性质，可以得到比较复杂强大的模型。

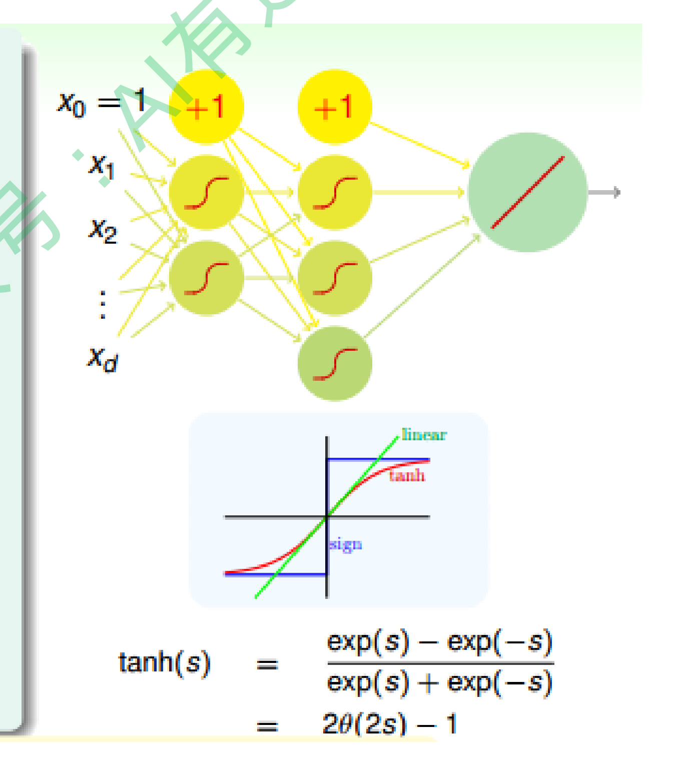
顺便提一下，tanh(x)函数与sigmoid函数存在下列关系：

$$tanh(s)=2\theta(2s)-1$$

其中，

$$\theta(s)=\frac{1}{1+exp(-s)}$$

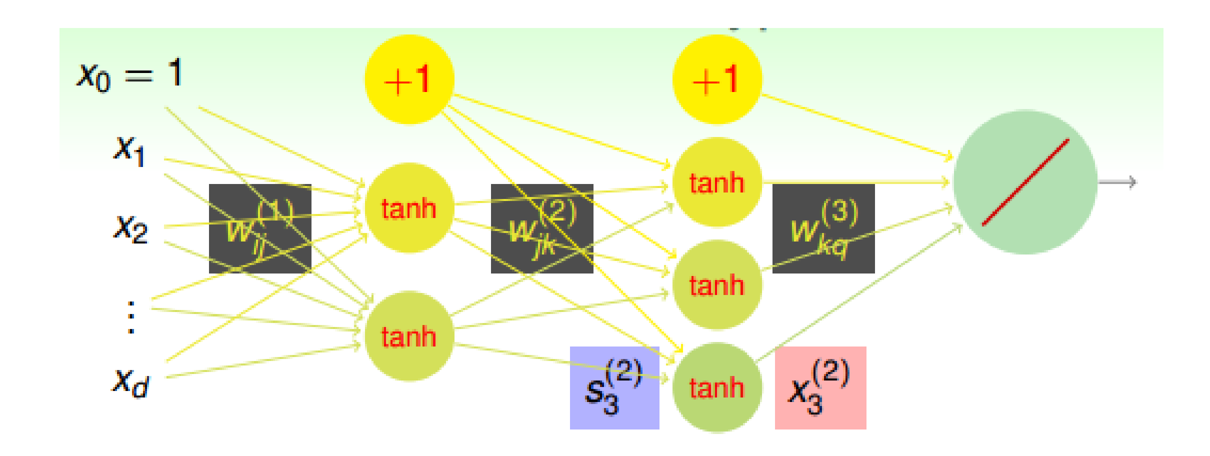
- : **transformation** function of score (signal) s
- any transformation?
  - : whole network linear & thus **less useful**
  - : discrete & thus **hard to optimize** for **w**
- popular choice of transformation: $= \tanh(s)$
  - ‘analog’ approximation of : **easier to optimize**
  - somewhat **closer to biological** neuron
  - **not that new! :-)**



那么，接下来我们就使用tanh函数作为神经网络中间层的transformation function，所有的数学推导也基于此。实际应用中，可以选择其它的transformation function，不同的transformation function，则有不同的推导过程。

下面我们将仔细来看看Neural Network Hypothesis的结构。如下图所示，该神经网络左边是输入层，中间两层是隐藏层，右边是输出层。整体上来说，我们设定输入层为第0层，然后往右分别是第一层、第二层，输出层即为第3层。





Neural Network Hypothesis中，$d^{(0)}, d^{(1)}, \cdots, d^{(L)}$分别表示神经网络的第几层，其中L为总层数。例如上图所示的是3层神经网络，L=3。我们先来看看每一层的权重$w_{ij}^{(l)}$，上标l满足$1 \le l \le L$，表示是位于哪一层。下标i满足$0 \le i \le d^{(l-1)}$，表示前一层输出的个数加上bias项（常数项）。下标满足$1 \le j \le d^{(l)}$，表示该层节点的个数（不包括bias项）。

对于每层的分数score，它的表达式为：

$$s_j^{(l)} = \sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}$$

对于每层的transformation function，它的表达式为：

$$x_j^{(l)} = \begin{cases} tanh(s_j^{(l)}), & if\ l < L \\ s_j^{(l)}, & if\ l = L \end{cases}$$

因为是regression模型，所以在输出层（l=L）直接得到$x_j^{(l)} = s_j^{(l)}$。

$d^{(0)}$-$d^{(1)}$-$d^{(2)}$-$\cdots$-$d^{(L)}$ Neural Network (NNet)

$w_{ij}^{(\ell)}$: $\begin{cases} 1 \le \ell \le L & \text{layers} \\ 0 \le i \le d^{(\ell-1)} & \text{inputs} \\ 1 \le j \le d^{(\ell)} & \text{outputs} \end{cases}$, score $s_j^{(\ell)} = \sum_{i=0}^{d^{(\ell-1)}} w_{ij}^{(\ell)} x_i^{(\ell-1)}$,

transformed $x_j^{(\ell)} = \begin{cases} \tanh\left(s_j^{(\ell)}\right) & \text{if } \ell < L \\ s_j^{(\ell)} & \text{if } \ell = L \end{cases}$

介绍完Neural Network Hypothesis的结构之后，我们来研究下这种算法结构到底有什么实际的物理意义。还是看上面的神经网络结构图，每一层输入到输出的运算过程，实际上都是一种transformation，而转换的关键在于每个权重值$w_{ij}^{(l)}$。每层网络利用输入x和权重w的乘积，在经过tanh函数，得到该层的输出，从左到右，一层一层地进

行。其中，很明显，x和w的乘积$\sum_{i=0}^{d^{(l-1)}} w_{ij}^{(l)} x_i^{(l-1)}$越大，那么tanh(wx)就会越接近1，表明这种transformation效果越好。再想一下，w和x是两个向量，乘积越大，表明两个向量内积越大，越接近平行，则表明w和x有模式上的相似性。从而，更进一步说明了如果每一层的输入向量x和权重向量w具有模式上的相似性，比较接近平行，那么transformation的效果就比较好，就能得到表现良好的神经网络模型。也就是说，神经网络训练的核心就是pattern extraction，即从数据中找到数据本身蕴含的模式和规律。通过一层一层找到这些模式，找到与输入向量x最契合的权重向量w，最后再由G输出结果。

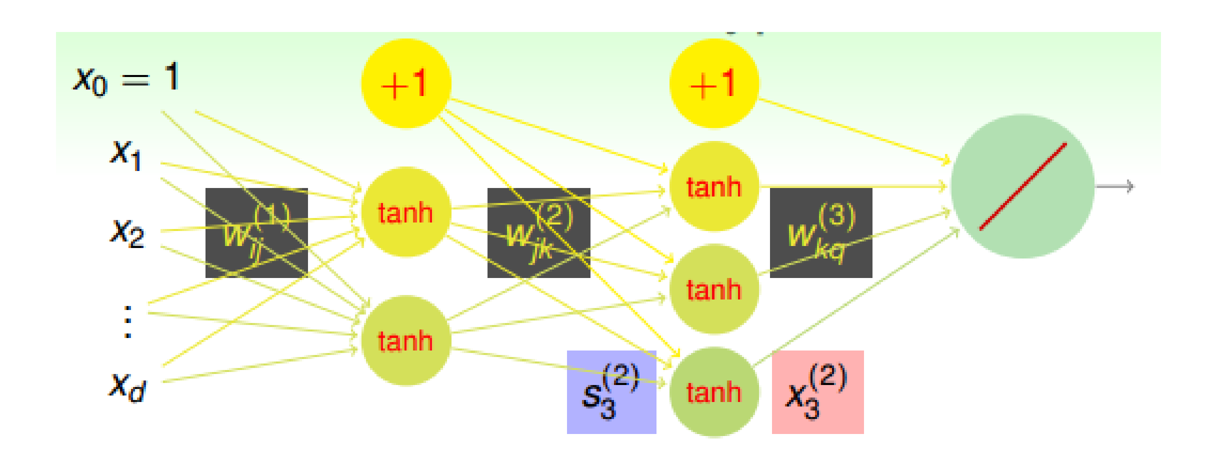
- each layer: **transformation** to be **learned** from data
- $\phi^{(\ell)}(\mathbf{x}) = \tanh\left(\begin{bmatrix} \sum_{i=0}^{d^{(\ell-1)}} w_{i1}^{(\ell)} x_i^{(\ell-1)} \\ \vdots \end{bmatrix}\right)$

—whether **x** 'matches' weight vectors in pattern

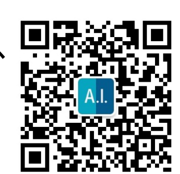
NNet: **pattern extraction** with layers of connection weights

Neural Network Learning

我们已经介绍了Neural Network Hypothesis的结构和算法流程。确定网络结构其实就是确定各层的权重值$w_{ij}^{(l)}$。那如何根据已有的样本数据，找到最佳的权重$w_{ij}^{(l)}$使error最小化呢？下面我们将详细推导。



首先，我们的目标是找到最佳的$w_{ij}^{(l)}$让$E_{in}(\{w_{ij}^{(l)}\})$最小化。如果只有一层隐藏层，就相当于是aggregation of perceptrons。可以使用我们上节课介绍的gradient boosting算法来一个一个确定隐藏层每个神经元的权重，输入层到隐藏层的权重可以



通过C&RT算法计算的到。这不是神经网络常用的算法。如果隐藏层个数有两个或者更多，那么aggregation of perceptrons的方法就行不通了。就要考虑使用其它方法。

根据error function的思想，从输出层来看，我们可以得到每个样本神经网络预测值与实际值之间的squared error：$e_n=(y_n-NNet(x_n))^2$，这是单个样本点的error。那么，我们只要能建立e_n与每个权重$w_{ij}^{(l)}$的函数关系，就可以利用GD或SGD算法对$w_{ij}^{(l)}$求偏微分，不断迭代优化$w_{ij}^{(l)}$值，最终得到使e_n最小时对应的$w_{ij}^{(l)}$。

- goal: learning all $\{w_{ij}^{(\ell)}\}$ to **minimize** $E_{in}\left(\{w_{ij}^{(\ell)}\}\right)$
- one hidden layer: simply **aggregation of perceptrons**
  —**gradient boosting** to determine hidden neuron one by one
- multiple hidden layers? **not easy**
- let $e_n=(y_n-\text{NNet}(\mathbf{x}_n))^2$:
  can apply **(stochastic) GD** after computing $\frac{\partial e_n}{\partial w_{ij}^{(\ell)}}$!

为了建立e_n与各层权重$w_{ij}^{(l)}$的函数关系，求出e_n对$w_{ij}^{(l)}$的偏导数$\frac{\partial e_n}{w_{ij}^{(l)}}$，我们先来看输出层如何计算$\frac{\partial e_n}{w_{i1}^{(L)}}$。$e_n$与$w_{i1}^{(L)}$的函数关系为：

$$e_n=(y_n-\text{NNet}(\mathbf{x}_n))^2=\left(y_n-s_1^{(L)}\right)^2=\left(y_n-\sum_{i=0}^{d^{(L-1)}}w_{i1}^{(L)}x_i^{(L-1)}\right)^2$$

计算e_n对$w_{i1}^{(L)}$的偏导数，得到：



specially (output layer)
$(0 \leq i \leq d^{(L-1)})$

$$
\begin{aligned}
&\frac{\partial e_n}{\partial w_{i1}^{(L)}} \\
= \ &\frac{\partial e_n}{\partial s_1^{(L)}} \cdot \frac{\partial s_1^{(L)}}{\partial w_{i1}^{(L)}} \\
= \ &-2\left(y_n - s_1^{(L)}\right) \cdot \left(x_i^{(L-1)}\right)
\end{aligned}
$$

以上是输出层求偏导的结果。如果是其它层，即$l \neq L$，偏导计算可以写成如下形式：

generally $(1 \leq \ell < L)$
$(0 \leq i \leq d^{(\ell-1)}; 1 \leq j \leq d^{(\ell)})$

$$
\begin{aligned}
&\frac{\partial e_n}{\partial w_{ij}^{(\ell)}} \\
= \ &\frac{\partial e_n}{\partial s_j^{(\ell)}} \cdot \frac{\partial s_j^{(\ell)}}{\partial w_{ij}^{(\ell)}} \\
= \ &\delta_j^{(\ell)} \cdot \left(x_i^{(\ell-1)}\right)
\end{aligned}
$$

上述推导中，令e_n与第l层第j个神经元的分数$s_j^{(l)}$的偏导数记为$\delta_j^{(l)}$。即：

$$\frac{\partial e_n}{\partial s_j^{(l)}} = \delta_j^{(l)}$$

当$l = L$时，$\delta_1^{(L)} = -2(y_n - s_1^{(L)})$；当$l \neq L$时，$\delta_j^{(l)}$是未知的，下面我们将进行运算推导，看看不同层之间的$\delta_j^{(l)}$是否有递推关系。



$$s_j^{(\ell)} \overset{\tanh}{\Longrightarrow} x_j^{(\ell)} \overset{w_{jk}^{(\ell+1)}}{\Longrightarrow} \begin{bmatrix} s_1^{(\ell+1)} \\ \vdots \\ s_k^{(\ell+1)} \\ \vdots \end{bmatrix} \Longrightarrow \cdots \Longrightarrow e_n$$

如上图所示，第l层第j个神经元的分数$s_j^{(l)}$经过tanh函数，得到该层输出$x_j^{(l)}$，再与下一层权重$w_{jk}^{(l+1)}$相乘，得到第l+1层的分数$s_j^{(l+1)}$，直到最后的输出层e_n。

那么，利用上面$s_j^{(l)}$到$s_j^{(l+1)}$这样的递推关系，我们可以对偏导数$\delta_j^{(l)}$做一些中间变量替换处理，得到如下表达式：

$$\begin{aligned} \delta_j^{(\ell)} = \frac{\partial e_n}{\partial s_j^{(\ell)}} &= \sum_{k=1}^{d^{(\ell+1)}} \frac{\partial e_n}{\partial s_k^{(\ell+1)}} \frac{\partial s_k^{(\ell+1)}}{\partial x_j^{(\ell)}} \frac{\partial x_j^{(\ell)}}{\partial s_j^{(\ell)}} \\ &= \sum_k \left(\delta_k^{(\ell+1)}\right) \left(w_{jk}^{(\ell+1)}\right) \left(\tanh'\left(s_j^{(\ell)}\right)\right) \end{aligned}$$

值得一提的是，上式中有个求和项，其中k表示下一层即l+1层神经元的个数。表明l层的$s_j^{(l)}$与l+1层的所有$s_k^{(l+1)}$都有关系。因为$s_j^{(l)}$参与到每个$s_k^{(l+1)}$的运算中了。

这样，我们得到了$\delta_j^{(l)}$与$\delta_k^{(l)}$的递推关系。也就是说如果知道了$\delta_k^{(l)}$的值，就能推导出$\delta_j^{(l)}$的值。而最后一层，即输出层的$\delta_1^{(L)} = -2(y_n - s_1^{(L)})$，那么就能一层一层往前推导，得到每一层的$\delta_j^{(l)}$，从而可以计算出$e_n$对各个$w_{ij}^{(l)}$的偏导数$\frac{\partial e_n}{w_{ij}^{(l)}}$。计算完偏微分之后，就可以使用GD或SGD算法进行权重的迭代优化，最终得到最优解。

神经网络中，这种从后往前的推导方法称为Backpropagation Algorithm，即我们常常听到的BP神经网络算法。它的算法流程如下所示：



Backprop on NNet

initialize all weights $w_{ij}^{(\ell)}$
for $t = 0, 1, \ldots, T$

1. stochastic: randomly pick $n \in \{1, 2, \cdots, N\}$
2. forward: compute all $x_i^{(\ell)}$ with $\mathbf{x}^{(0)} = \mathbf{x}_n$
3. backward: compute all $\delta_j^{(\ell)}$ subject to $\mathbf{x}^{(0)} = \mathbf{x}_n$
4. gradient descent: $w_{ij}^{(\ell)} \leftarrow w_{ij}^{(\ell)} - \eta x_i^{(\ell-1)} \delta_j^{(\ell)}$

return $g_{\text{NNET}}(\mathbf{x}) = \left(\cdots \tanh\left(\sum_j w_{jk}^{(2)} \cdot \tanh\left(\sum_i w_{ij}^{(1)} x_i\right)\right)\right)$

上面采用的是SGD的方法，即每次迭代更新时只取一个点，这种做法一般不够稳定。所以通常会采用mini-batch的方法，即每次选取一些数据，例如$\frac{N}{10}$，来进行训练，最后求平均值更新权重w。这种做法的实际效果会比较好一些。

Optimization and Regularization

经过以上的分析和推导，我们知道神经网络优化的目标就是让$E_{in}(w)$最小化。本节课我们采用error measure是squared error，当然也可以采用其它的错误衡量方式，只要在推导上做稍稍修改就可以了，此处不再赘述。

$$E_{\text{in}}(\mathbf{w}) = \frac{1}{N}\sum_{n=1}^{N} \text{err}\left(\left(\cdots \tanh\left(\sum_j w_{jk}^{(2)} \cdot \tanh\left(\sum_i w_{ij}^{(1)} x_{n,i}\right)\right)\right), y_n\right)$$

下面我们将主要分析神经网络的优化问题。由于神经网络由输入层、多个隐藏层、输出层构成，结构是比较复杂的非线性模型，因此$E_{in}(w)$可能有许多局部最小值，是non-convex的，找到全局最小值（globalminimum）就会困难许多。而我们使用GD或SGD算法得到的很可能就是局部最小值（local minimum）。

基于这个问题，不同的初始值权重$w_{ij}^{(l)}$通常会得到不同的local minimum。也就是说最终的输出G与初始权重$w_{ij}^{(l)}$有很大的关系。在选取$w_{ij}^{(l)}$上有个技巧，就是通常选择比较小的值，而且最好是随机random选择。这是因为，如果权重$w_{ij}^{(l)}$很大，那么根据tanh函数，得到的值会分布在两侧比较平缓的位置（类似于饱和saturation），这时候梯度很小，每次迭代权重可能只有微弱的变化，很难在全局上快速得到最优解。而随机选择的原因是通常对权重$w_{ij}^{(l)}$如何选择没有先验经验，只能通过random，从普遍概



率上选择初始值，随机性避免了人为因素的干预，可以说更有可能经过迭代优化得到全局最优解。

- generally non-convex when multiple hidden layers
  - not easy to reach global minimum
  - GD/SGD with backprop only gives local minimum
- different initial $w_{ij}^{(\ell)}$ $\Longrightarrow$ different local minimum
  - somewhat 'sensitive' to initial weights
  - large weights $\Longrightarrow$ saturate (small gradient)
  - advice: try some random & small ones

下面从理论上看一下神经网络模型的VC Dimension。对于tanh这样的transfer function，其对应的整个模型的复杂度$d_{vc}=O(VD)$。其中V是神经网络中神经元的个数（不包括bias点）,D表示所有权值的数量。所以，如果V足够大的时候，VC Dimension也会非常大，这样神经网络可以训练出非常复杂的模型。但同时也可能会造成过拟合overfitting。所以，神经网络中神经元的数量V不能太大。

为了防止神经网络过拟合，一个常用的方法就是使用regularization。之前我们就介绍过可以在error function中加入一个regularizer，例如熟悉的L2 regularizer $\Omega(w)$：

$$\Omega(w)=\sum(w_{ij}^{(l)})^2$$

但是，使用L2 regularizer 有一个缺点，就是它使每个权重进行等比例缩小（shrink）。也就是说大的权重缩小程度较大，小的权重缩小程度较小。这会带来一个问题，就是等比例缩小很难得到值为零的权重。而我们恰恰希望某些权重$w_{ij}^{(l)}=0$，即权重的解是松散（sparse）的。因为这样能有效减少VC Dimension，从而减小模型复杂度，防止过拟合发生。

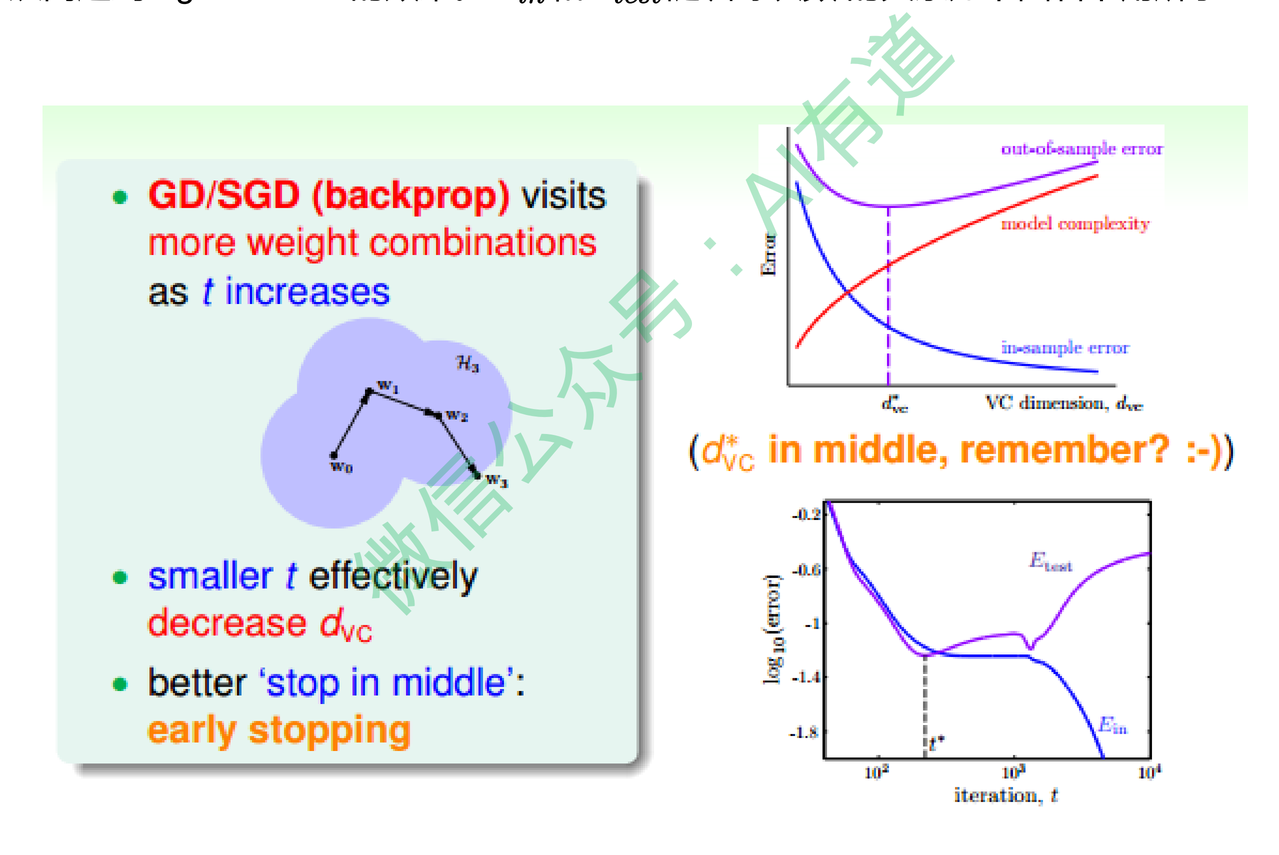
那么为了得到sparse解，有什么方法呢？我们之前就介绍过可以使用L1 regularizer：$\sum|wij^{(l)}|$，但是这种做法存在一个缺点，就是包含绝对值不容易微分。除此之外，另外一种比较常用的方法就是使用weight-elimination regularizer。weight-elimination regularizer类似于L2 regularizer，只不过是在L2 regularizer上做了尺度的缩小，这样能使large weight和small weight都能得到同等程度的缩小，从而让更多权重最终为零。weight-elimination regularizer的表达式如下：

$$\sum\frac{(w_{ij}^{(l)})^2}{1+(w_{ij}^{(l)})^2}$$



- 'shrink' weights:
  large weight → large shrink; small weight → small shrink
- want $w_{ij}^{(\ell)} = 0$ (sparse) to effectively **decrease** d_{VC}
  - L1 regularizer: $\sum \left| w_{ij}^{(\ell)} \right|$, but **not differentiable**
  - weight-elimination ('scaled' L2) regularizer:
    large weight → median shrink; small weight → median shrink

除了weight-elimination regularizer之外，还有另外一个很有效的regularization的方法，就是Early Stopping。简而言之，就是神经网络训练的次数t不能太多。因为，t太大的时候，相当于给模型寻找最优值更多的可能性，模型更复杂，VC Dimension增大，可能会overfitting。而t不太大时，能有效减少VC Dimension，降低模型复杂度，从而起到regularization的效果。E_{in}和E_{test}随训练次数t的关系如下图右下角所示：



那么，如何选择最佳的训练次数t呢？可以使用validation进行验证选择。

总结

本节课主要介绍了Neural Network模型。首先，我们通过使用一层甚至多层的perceptrons来获得更复杂的非线性模型。神经网络的每个神经元都相当于一个Neural Network Hypothesis，训练的本质就是在每一层网络上进行pattern extraction，找到最合适的权重$w_{ij}^{(l)}$，最终得到最佳的G。本课程以regression模型为例，最终的G是线



性模型，而中间各层均采用tanh函数作为transform function。计算权重$w_{ij}^{(l)}$的方法就是采用GD或者SGD，通过Backpropagation算法，不断更新优化权重值，最终使得$E_{in}(w)$最小化，即完成了整个神经网络的训练过程。最后，我们提到了神经网络的可以使用一些regularization来防止模型过拟合。这些方法包括随机选择较小的权重初始值，使用weight-elimination regularizer或者early stopping等。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程





林轩田《机器学习技法》课程笔记13 -- Deep Learning

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了神经网络Neural Network。神经网络是由一层一层的神经元构成，其作用就是帮助提取原始数据中的模式即特征，简称为pattern feature extraction。神经网络模型的关键是计算出每个神经元的权重，方法就是使用Backpropagation算法，利用GD/SGD，得到每个权重的最优解。本节课我们将继续对神经网络进行深入研究，并介绍层数更多、神经元个数更多、模型更复杂的神经网络模型，即深度学习模型。

Deep Neural Network

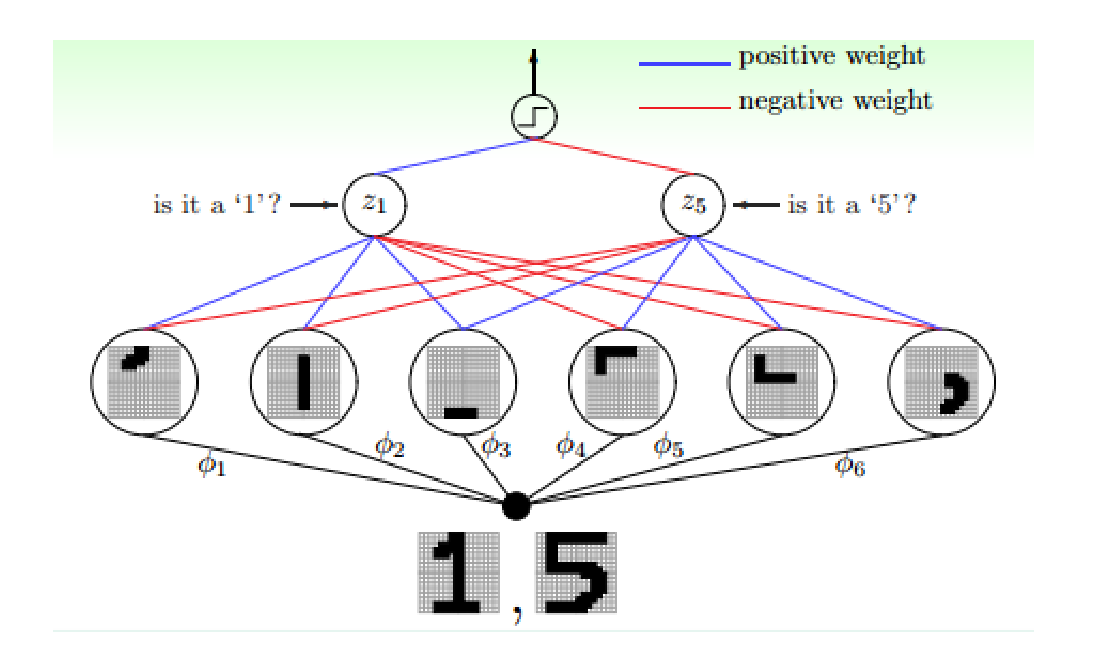
总的来说，根据神经网络模型的层数、神经元个数、模型复杂度不同，大致可分为两类：Shallow Neural Networks和Deep Neural Networks。上节课介绍的神经网络模型层数较少，属于Shallow Neural Networks，而本节课将着重介绍Deep Neural Networks。首先，比较一下二者之间的优缺点有哪些：

Shallow NNet	Deep NNet
• more efficient to train (◯)	• challenging to train (×)
• simpler structural decisions (◯)	• sophisticated structural decisions (×)
• theoretically powerful enough (◯)	• 'arbitrarily' powerful (◯)
	• more 'meaningful'? (see next slide)

值得一提的是，近些年来，deep learning越来越火，尤其在电脑视觉和语音识别等领域都有非常广泛的应用。原因在于一层一层的神经网络有助于提取图像或者语音的一些物理特征，即pattern feature extraction，从而帮助人们掌握这些问题的本质，建立准确的模型。

下面举个例子，来看一下深度学习是如何提取出问题潜在的特征从而建立准确的模型的。如下图所示，这是一个手写识别的问题，简单地识别数字1和数字5。





如何进行准确的手写识别呢？我们可以将写上数字的图片分解提取出一块一块不同部位的特征。例如左边三幅图每张图代表了数字1的某个部位的特征，三幅图片组合起来就是完整的数字1。右边四幅图也是一样，每张图代表了数字5的某个部位的特征，五幅图组合起来就是完整的数字5。对计算机来说，图片由许多像素点组成。要达到识别的目的，每层神经网络从原始像素中提取出更复杂的特征，再由这些特征对图片内容进行匹配和识别。层数越多，提取特征的个数和深度就越大，同时解决复杂问题的能量就越强，其中每一层都具有相应的物理意义。以上就是深度学习的作用和意义。

深度学习很强大，同时它也面临很多挑战和困难：

- **difficult structural decisions**
- **high model complexity**
- **hard optimization problem**
- **huge computational complexity**

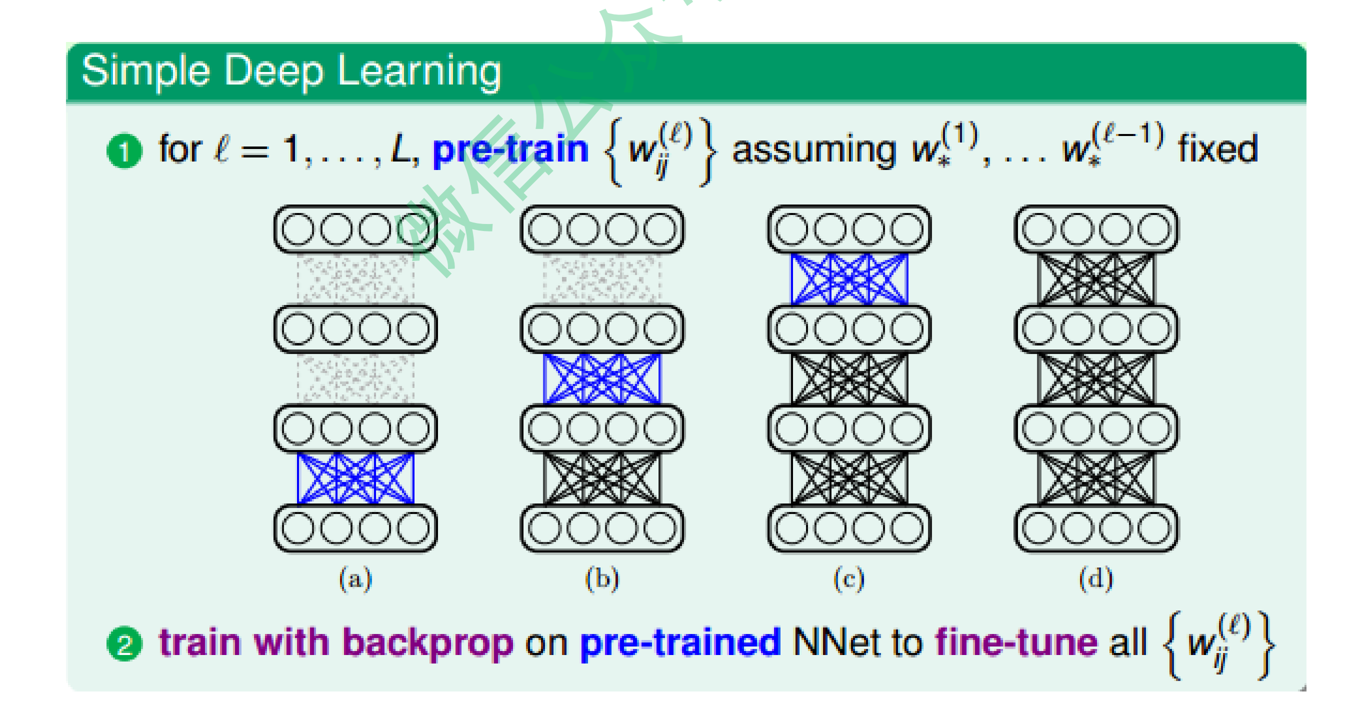
面对以上深度学习的4个困难，有相应的技术和解决的办法：



- difficult **structural decisions**:
  - subjective with **domain knowledge**: like **convolutional NNet** for images
- high **model complexity**:
  - no big worries if **big enough data**
  - **regularization** towards noise-tolerant: like
    - **dropout** (tolerant when network corrupted)
    - **denoising** (tolerant when input corrupted)
- hard **optimization problem**:
  - **careful initialization** to avoid bad local minimum: called **pre-training**
- huge **computational complexity** (worsen with **big data**):
  - novel hardware/architecture: like **mini-batch with GPU**

其中，最关键的技术就是regularization和initialization。

深度学习中，权重的初始化选择很重要，好的初始值能够帮助避免出现局部最优解的出现。常用的方法就是pre-train，即先权重进行初始值的选择，选择之后再使用backprop算法训练模型，得到最佳的权重值。在接下来的部分，我们将重点研究pre-training的方法。



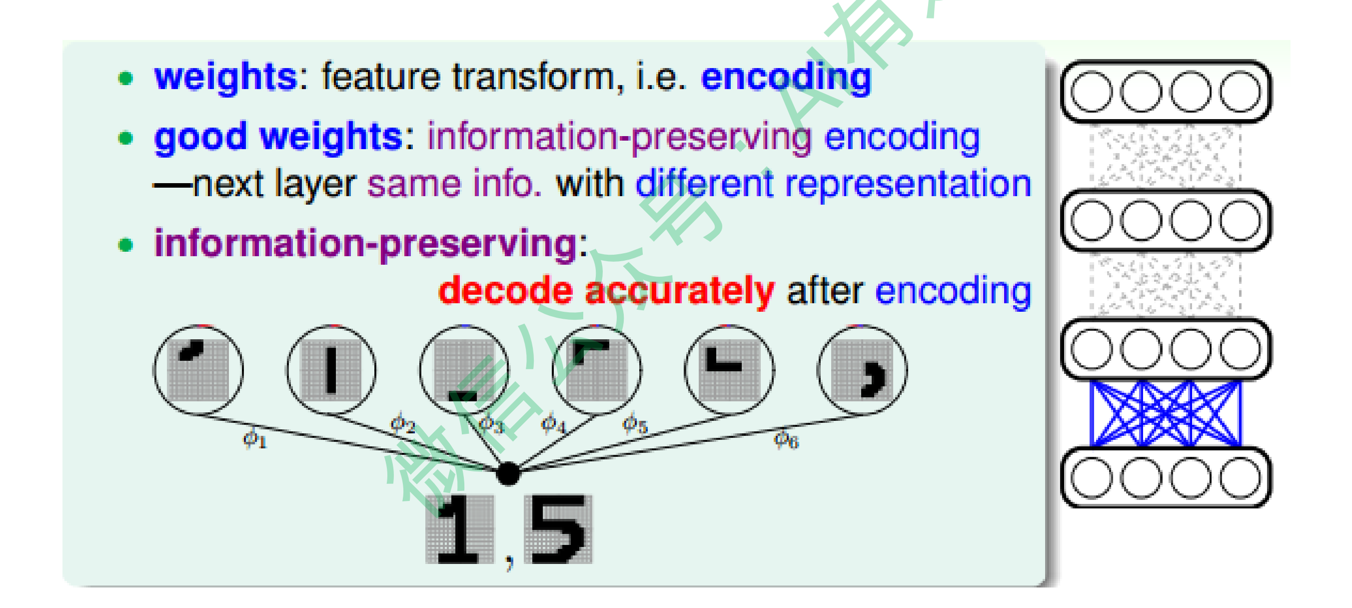
Autoencoder

我们已经介绍了深度学习的架构，那么从算法模型上来说，如何进行pre-training，得到较好的权重初始值呢？首先，我们来看看，权重是什么？神经网络模型中，权重代



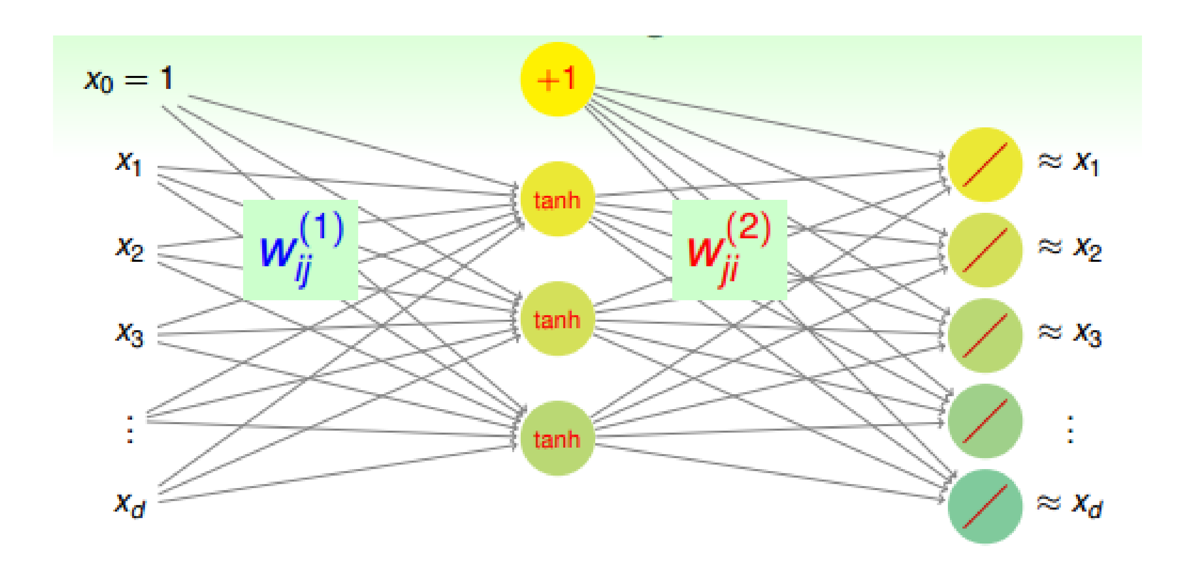
表了特征转换（feature transform）。从另一个方面也可以说，权重表示一种编码（encoding），就是把数据编码成另外一些数据来表示。因为神经网络是一层一层进行的，有先后顺序，所以就单一层来看，好的权重初始值应该是尽可能地包含了该层输入数据的所有特征，即类似于information-preserving encoding。也就是说，能够把第i层的输入数据的特征传输到第i+1层，再把第i+1层的输入数据的特征传输到第i+2层，一层一层进行下去。这样，每层的权重初始值起到了对该层输入数据的编码作用，能够最大限度地保持其特征。

举个例子，上一小节我们讲了简单的手写识别的例子。从原始的一张像素图片转换到分解的不同笔画特征，那么反过来，这几个笔画特征也可以组合成原来的数字。这种可逆的转换被称为information-preserving，即转换后的特征保留了原输入的特征，而且转换是可逆的。这正是pre-train希望做到的，通过encoding将输入转换为一些特征，而这些特征又可以复原原输入x，实现information-preserving。所以，pre-training得到的权重初始值就应该满足这样的information-preserving特性。



如何在pre-training中得到这样的权重初始值（即转换特征）呢？方法是建立一个简单的三层神经网络（一个输入层、一个隐藏层、一个输出层），如下图所示。





该神经网络中，输入层是原始数据（即待pre-training的数据），经过权重$W_{ij}^{(1)}$得到隐藏层的输出为原始数据新的表达方式（即转换特征）。这些转换特征再经过权重$W_{ji}^{(2)}$得到输出层，输出层的结果要求跟原始数据类似，即输入层和输出层是近似相等的。整个网络是$d-\check{d}-d$ NNet结构。其核心在于“重构性”，从输入层到隐藏层实现特征转换，从隐藏层到输出层实现重构，满足上文所说的information-preserving的特性。这种结构的神经网络我们称之为autoencoder，输入层到隐藏层对应编码，而隐藏层到输出层对应解码。其中，$W_{ij}^{(1)}$表示编码权重，而$W_{ji}^{(2)}$表示解码权重。整个过程类似于在学习如何近似逼近identity function。

- **autoencoder**: d—$\tilde{d}$—d NNet with goal $g_i(\mathbf{x}) \approx x_i$
  —learning to **approximate identity function**
- $w_{ij}^{(1)}$: encoding weights; $w_{ji}^{(2)}$: decoding weights

那么为什么要使用这样的结构来逼近identity function，有什么好处呢？首先对于监督式学习（supervised learning），这种$d-\check{d}-d$的NNet结构中含有隐藏层。隐藏层的输出实际上就是对原始数据合理的特征转换$\phi(x)$，例如手写识别中隐藏层分解的各个笔画，包含了有用的信息。这样就可以从数据中学习得到一些有用的具有代表性的信息。然后，对于非监督式学习（unsupervised learning），autoencoder也可以用来做density estimation。如果网络最终的输出$g(x) \approx x$，则表示密度较大；如果g(x)与x相差甚远，则表示密度较小。也就是说可以根据g(x)与x的接近程度来估计测试数据是落在密度较大的地方还是密度较小的地方。这种方法同样适用于outlier detection，异常检测。这样就可以从数据中学习得到一些典型的具有代表性的信息，找出哪些是典型资料，哪些不是典型资料。所以说，通过autoencoder不断逼近identity function，对监督式学习和非监督式学习都具有深刻的物理意义和非常广泛的应用。



if $\mathbf{g}(\mathbf{x}) \approx \mathbf{x}$ using some **hidden** structures on the **observed data $\mathbf{x}_n$**

- for supervised learning:
  - hidden structure (essence) of $\mathbf{x}$ can be used as reasonable transform $\Phi(\mathbf{x})$

  —learning **'informative' representation** of data
- for unsupervised learning:
  - density estimation: larger (structure match) when $\mathbf{g}(\mathbf{x}) \approx \mathbf{x}$
  - outlier detection: those $\mathbf{x}$ where $\mathbf{g}(\mathbf{x}) \not\approx \mathbf{x}$

  —learning **'typical' representation** of data

其实，对于autoencoder来说，我们更关心的是网络中间隐藏层，即原始数据的特征转换以及特征转换的编码权重$W_{ij}^{(1)}$。

Basic Autoencoder一般采用$d-\check{d}-d$的NNet结构，对应的error function是squared error，即$\sum_{i=1}^{d}(g_i(x)-x_i)^2$。

basic **autoencoder**:

$$d—\tilde{d}—d \text{ NNet with error function } \sum_{i=1}^{d}(g_i(\mathbf{x})-x_i)^2$$

basic autoencoder在结构上比较简单，只有三层网络，容易训练和优化。各层之间的神经元数量上，通常限定$\check{d}<d$，便于数据编码。数据集可表示为：$\{(x_1,y_1=x_1),(x_2,y_2=x_2),\cdots,(x_N,y_N=x_N)\}$，即输入输出都是x，可以看成是非监督式学习。一个重要的限制条件是$W_{ij}^{(1)}=W_{ji}^{(2)}$，即编码权重与解码权重相同。这起到了regularization的作用，但是会让计算复杂一些。

- backprop easily applies; **shallow** and easy to train
- usually $\tilde{d} < d$: **compressed** representation
- data: $\{(\mathbf{x}_1, \mathbf{y}_1 = \mathbf{x}_1), (\mathbf{x}_2, \mathbf{y}_2 = \mathbf{x}_2), \ldots, (\mathbf{x}_N, \mathbf{y}_N = \mathbf{x}_N)\}$

  —often categorized as **unsupervised learning technique**
- sometimes constrain $w_{ij}^{(1)} = w_{ji}^{(2)}$ as **regularization**

  —more **sophisticated** in calculating gradient

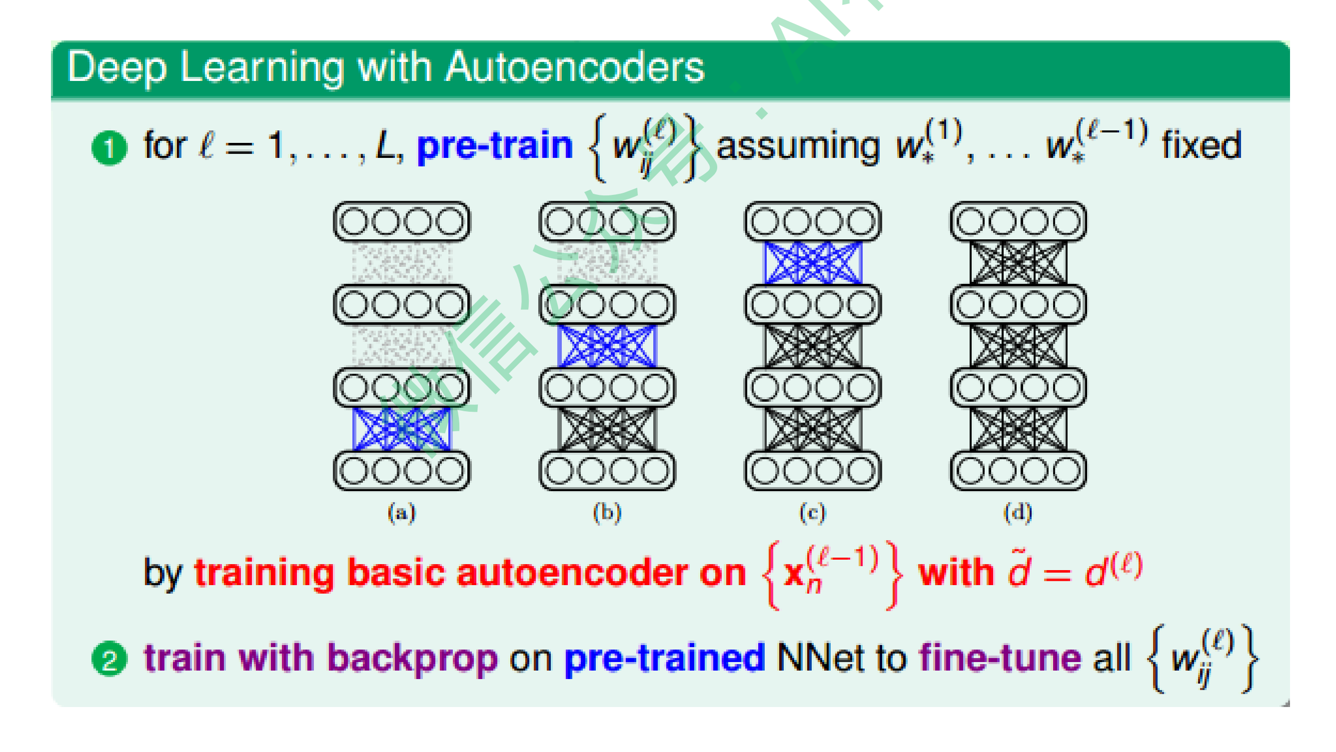
以上就是basic autoencoder的结构和一些限定条件。深度学习中，basic autoencoder的过程也就对应着pre-training的过程，使用这种方法，对无label的原始数据进行编码



和解码，得到的编码权重$W_{ij}^{(1)}$就可以作为pre-trained的比较不错的初始化权重，也就是作为深度学习中层与层之间的初始化权重。

basic **autoencoder** in basic deep learning:
$\left\{w_{ij}^{(1)}\right\}$ taken as shallowly pre-trained weights

我们在本节课第一部分就说了深度学习中非常重要的一步就是pre-training，即权重初始化，而autoencoder可以作为pre-training的一个合理方法。Pre-training的整个过程是：首先，autoencoder会对深度学习网络第一层（即原始输入）进行编码和解码，得到编码权重$W_{ij}^{(1)}$，作为网络第一层到第二层的的初始化权重；然后再对网络第二层进行编码和解码，得到编码权重$W_{ij}^{(1)}$，作为网络第二层到第三层的初始化权重，以此类推，直到深度学习网络中所有层与层之间都得到初始化权重。值得注意的是，对于l-1层的网络$\{x_n^{(l-1)}\}$，autoencoder中的$\check{d}$应与下一层（即l层）的神经元个数相同。

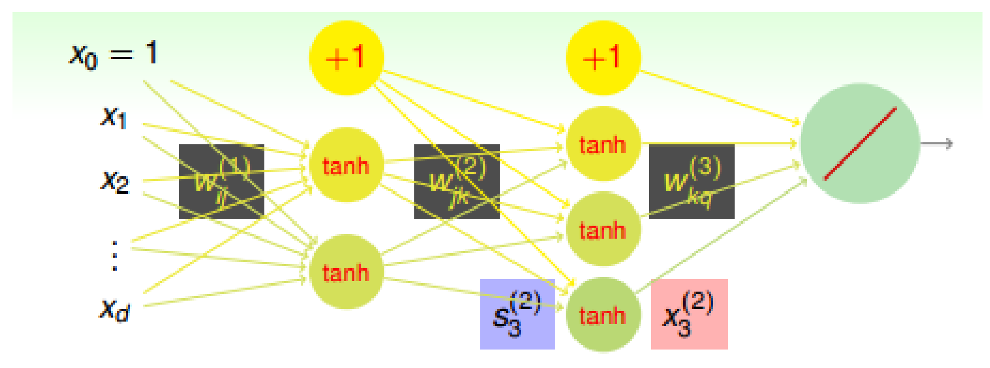


当然，除了basic autoencoder之外还有许多其它表现不错的pre-training方法。这些方法大都采用不同的结构和正则化技巧来得到不同的'fancier' autoencoders，这里不再赘述。

Denoising Autoencoder

上一部分，我们使用autoencoder解决了deep learning中pre-training的问题。接下来，我们将讨论deep learning中有什么样的regularization方式来控制模型的复杂度。





watch out for overfitting, remember? :-)

由于深度学习网络中神经元和权重的个数非常多，相应的模型复杂度就会很大，因此，regularization非常必要。之前我门也介绍过一些regularization的方法，包括：

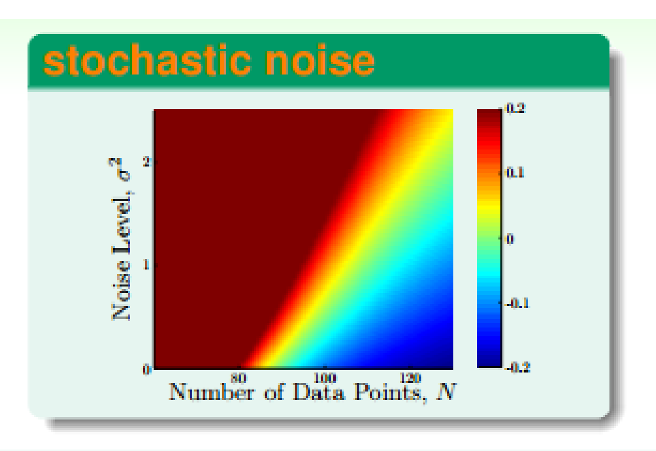
- **structural decisions/constraints**
- **weight decay or weight elimination regularizers**
- **early stopping**

high **model complexity**: **regularization** needed
- structural decisions/**constraints**
- weight decay or weight elimination **regularizers**
- **early stopping**

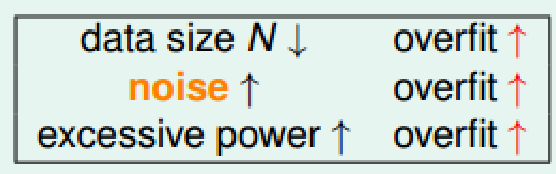
下面我们将介绍另外一种regularization的方式，它在deep learning和autoencoder中都有很好的效果。

首先我们来复习一下之前介绍的overfitting产生的原因有哪些。如下图所示，我们知道overfitting与样本数量、噪声大小都有关系，数据量减少或者noise增大都会造成overfitting。如果数据量是固定的，那么noise的影响就非常大，此时，实现regularization的一个方法就是消除noise的影响。





reasons of serious overfitting:



data size $N \downarrow$	overfit $\uparrow$
noise $\uparrow$	overfit $\uparrow$
excessive power $\uparrow$	overfit $\uparrow$

去除noise的一个简单方法就是对数据进行cleaning/pruning的操作。但是，这种方法通常比较麻烦，费时费力。此处，有一种比较“疯狂”的方法，就是往数据中添加一些noise。注意是添加noise！下面我们来解释这样做到底有什么作用。

- direct possibility: data cleaning/pruning, remember? :-)
- a wild possibility: adding noise to data?

这种做法的idea来自于如何建立一个健壮（robust）的autoencoder。在autoencoder中，编码解码后的输出g(x)会非常接近真实样本值x。此时，如果对原始输入加入一些noise，对于健壮的autoencoder，编码解码后的输出g(x)同样会与真实样本值x很接近。举个例子，手写识别中，通常情况下，写的很规范的数字1经过autoencoder后能够复原为数字1。如果原始图片数字1歪斜或加入噪声，经过autoencoder后应该仍然能够解码为数字1。这表明该autoencoder是robust的，一定程度上起到了抗噪声和regularization的作用，这正是我们希望看到的。

所以，这就引出了denoising autoencoder的概念。denoising autoencoder不仅能实现编码和解码的功能，还能起到去噪声、抗干扰的效果，即输入一些混入noise的数据，经过autoencoder之后能够得到较纯净的数据。这样，autoencoder的样本集为：

$$\{(\breve{x}_1, y_1 = x_1), (\breve{x}_2, y_2 = x_2), \cdots, (\breve{x}_N, y_N = x_N)\}$$

其中$\breve{x}_n = x_n + noise$，为混入噪声的样本，而$x_n$为纯净样本。

autoencoder训练的目的就是让$\breve{x}_n$经过编码解码后能够复原为纯净的样本x_n。那么，在deep learning的pre-training中，如果使用这种denoising autoencoder，不仅能从纯净的样本中编解码得到纯净的样本，还能从混入noise的样本中编解码得到纯净的样



本。这样得到的权重初始值更好，因为它具有更好的抗噪声能力，即健壮性好。实际应用中，denoising autoencoder非常有用，在训练过程中，输入混入人工noise，输出纯净信号，让模型本身具有抗噪声的效果，让模型健壮性更强，最关键的是起到了regularization的作用。

- idea: **robust** autoencoder should not only let $g(\mathbf{x}) \approx \mathbf{x}$ but also allow $g(\tilde{\mathbf{x}}) \approx \mathbf{x}$ even when $\tilde{\mathbf{x}}$ slightly different from $\mathbf{x}$
- **denoising** autoencoder:

  run basic autoencoder with data
  $\{(\tilde{\mathbf{x}}_1, \mathbf{y}_1 = \mathbf{x}_1), (\tilde{\mathbf{x}}_2, \mathbf{y}_2 = \mathbf{x}_2), \ldots, (\tilde{\mathbf{x}}_N, \mathbf{y}_N = \mathbf{x}_N)\}$,
  where $\tilde{\mathbf{x}}_n = \mathbf{x}_n +$ **artificial noise**

  —often used **instead of basic autoencoder** in deep learning
- useful for data/image processing: $g(\tilde{\mathbf{x}})$ a **denoised** version of $\tilde{\mathbf{x}}$
- effect: 'constrain/**regularize**' g towards **noise-tolerant** denoising

Principal Component Analysis

刚刚我们介绍的autoencoder是非线性的，因为其神经网络模型中包含了tanh()函数。这部分我们将介绍linear autoencoder。nonlinear autoencoder通常比较复杂，多应用于深度学习中；而linear autoencoder通常比较简单，我们熟知的主成分分析（Principal Component Analysis，PCA），其实跟linear autoencoder有很大的关系。

对于一个linear autoencoder，它的第k层输出不包含tanh()函数，可表示为：

$$h_k(x) = \sum_{j=0}^{\breve{d}} w_{jk}^{(2)} \left(\sum_{i=0}^{d} w_{ij}^{(1)} x_i\right)$$

其中，$w_{ij}^{(1)}$和$w_{jk}^{(2)}$分别是编码权重和解码权重。而且，有三个限制条件，分别是：

- **移除常数项x_0，让输入输出维度一致**
- **编码权重与解码权重一致：$w_{ij}^{(1)}=w_{jk}^{(2)}=w_{ij}**
- $\breve{d} < d$



linear hypothesis for k-th component $h_k(\mathbf{x}) = \sum_{j=0}^{\tilde{d}} w_{kj} \left(\sum_{i=1}^{d} w_{ij} x_i \right)$

consider three special conditions:

- **exclude** x_0: range of i **same** as range of k
- constrain $w_{ij}^{(1)} = w_{ji}^{(2)} = w_{ij}$: **regularization**
  —denote $\mathrm{W} = [w_{ij}]$ of size $d \times \tilde{d}$
- assume $\tilde{d} < d$: ensure **non-trivial** solution

这样，编码权重用W表示，维度是d x $\breve{d}$，解码权重用W^T表示。x的维度为d x 1。则linear autoencoder hypothesis可经过下式计算得到：

$$h(x) = WW^T x$$

其实，linear autoencoder hypothesis就应该近似于原始输入x的值，即h(x)=x。根据这个，我们可以写出它的error function：

$$E_{\text{in}}(\mathbf{h}) = E_{\text{in}}(\mathrm{W}) = \frac{1}{N} \sum_{n=1}^{N} \left\| \mathbf{x}_n - \mathrm{W}\mathrm{W}^T \mathbf{x}_n \right\|^2 \text{ with } d \times \tilde{d} \text{ matrix } \mathrm{W}$$

我们的目的是计算出$E_{in}(h)$最小化时对应的W。根据线性代数知识，首先进行特征值分解：

$$WW^T = V\Gamma V^T$$

其中WW^T是半正定矩阵。V矩阵满足$VV^T = V^T V = I_d$。Γ是对角矩阵，对角线上有不超过$\breve{d}$个非零值（即为1），即对角线零值个数大于等于$d - \breve{(d)}$。根据特征值分解的思想，我们可以把x_n进行类似分解：

$$x_n = VIV^T x_n$$

其中，I是单位矩阵，维度为dxd。这样，通过特征值分解我们就把对W的优化问题转换成对Γ和V的优化问题。



let's familiarize the problem with linear algebra (be brave! :-))

- eigen-decompose $WW^T = V\Gamma V^T$
  - $d \times d$ matrix V **orthogonal**: $VV^T = V^TV = I_d$
  - $d \times d$ matrix Γ **diagonal** with $\leq \tilde{d}$ non-zero
- $WW^T\mathbf{x}_n = V\Gamma V^T\mathbf{x}_n$
  - $V^T(\mathbf{x}_n)$: change of orthonormal basis (**rotate** or reflect)
  - $\Gamma(\cdots)$: set $\geq d - \tilde{d}$ components to 0, and **scale** others
  - $V(\cdots)$: reconstruct by coefficients and basis (**back-rotate**)
- $\mathbf{x}_n = VIV^T\mathbf{x}_n$: **rotate** and **back-rotate** cancel out

首先，我们来优化Γ值，表达式如下：

$$\min_V \min_\Gamma \frac{1}{N}\sum_{n=1}^{N}\left\|\underbrace{VIV^T\mathbf{x}_n}_{\mathbf{x}_n} - \underbrace{V\Gamma V^T\mathbf{x}_n}_{WW^T\mathbf{x}_n}\right\|^2$$

要求上式的最小化，可以转化为$(I-\Gamma)$越小越好，其结果对角线上零值越多越好，即I与Γ越接近越好。因为Γ的秩是小于等于$\breve{d}$的，Γ最多有$\breve{d}$个1。所以，Γ的最优解是其对角线上有$\breve{d}$个1。

- **back-rotate** not affecting length: ✗
- $\min_\Gamma \sum \|(I-\Gamma)(\text{some vector})\|^2$: **want many** 0 within $(I-\Gamma)$
- optimal diagonal Γ with rank $\leq \tilde{d}$:

$$\left\{\begin{array}{c}\tilde{d}\ \textbf{diagonal components 1}\\ \text{other components 0}\end{array}\right\} \Longrightarrow \text{without loss of gen.} \begin{bmatrix} I_{\tilde{d}} & 0 \\ 0 & 0 \end{bmatrix}$$

那么，Γ的最优解已经得出，表达式变成：

$$\text{next: } \min_V \sum_{n=1}^{N}\left\|\underbrace{\begin{bmatrix} 0 & 0 \\ 0 & I_{d-\tilde{d}} \end{bmatrix}}_{I-\text{optimal }\Gamma} V^T\mathbf{x}_n\right\|^2$$

这里的最小化问题似乎有点复杂，我们可以做一些转换，把它变成最大化问题求解，



转换后的表达式为：

$$\min_{\mathrm{V}} \sum_{n=1}^{N} \left\| \begin{bmatrix} 0 & 0 \\ 0 & \mathrm{I}_{d-\tilde{d}} \end{bmatrix} \mathrm{V}^T \mathbf{x}_n \right\|^2 \equiv \max_{\mathrm{V}} \sum_{n=1}^{N} \left\| \begin{bmatrix} \mathrm{I}_{\tilde{d}} & 0 \\ 0 & 0 \end{bmatrix} \mathrm{V}^T \mathbf{x}_n \right\|^2$$

当$\breve{d}=1$时，V^T中只有第一行v^T有用，最大化问题转化为：

$$max_v \sum_{n=1}^{N} v^T x_n x_n^T v \quad subject\ to\ v^T v = 1$$

引入拉格朗日因子λ，表达式的微分与条件微分应该是平行的，且由λ联系起来，即：

$$\sum_{n=1}^{N} x_n x_n^T v = \lambda v$$

根据线性代数的知识，很明显能够看出，v就是矩阵$X^T X$的特征向量，而λ就是相对应的特征值。我们要求的是最大值，所以最优解v就是矩阵$X^T X$最大特征值对应的特征向量。

当$\breve{d}>1$时，求解方法是类似的，最优解$\{v_j\}_{j=1}^{\breve{d}}$就是矩阵$X^T X$前$\breve{d}$大的特征值对应的$\breve{d}$个特征向量。

经过以上分析，我们得到了Γ和V的最优解。这就是linear autoencoder的编解码推导过程。

- $\tilde{d}=1$: only first row $\mathbf{v}^T$ of V^T matters

$$\max_{\mathbf{v}} \sum_{n=1}^{N} \mathbf{v}^T \mathbf{x}_n \mathbf{x}_n^T \mathbf{v} \text{ subject to } \mathbf{v}^T \mathbf{v} = 1$$

  - optimal $\mathbf{v}$ satisfies $\sum_{n=1}^{N} \mathbf{x}_n \mathbf{x}_n^T \mathbf{v} = \lambda \mathbf{v}$
    —using Lagrange multiplier λ, remember? :-)
  - optimal $\mathbf{v}$: 'topmost' eigenvector of $\mathrm{X}^T\mathrm{X}$
- general $\tilde{d}$: $\{\mathbf{v}_j\}_{j=1}^{\tilde{d}}$ 'topmost' eigenvectorS of $\mathrm{X}^T\mathrm{X}$
  —optimal $\{\mathbf{w}_j\} = \{\mathbf{v}_j$ with $[\![\gamma_j = 1]\!]\}$ = top eigenvectors

值得一提的是，linear autoencoder与PCA推导过程十分相似。但有一点不同的是，一般情况下，PCA会对原始数据x进行处理，即减去其平均值。这是为了在推导过程中的便利。这两种算法的计算流程大致如下：

Linear Autoencoder or PCA

1. let $\bar{\mathbf{x}} = \frac{1}{N}\sum_{n=1}^{N} \mathbf{x}_n$, and let $\mathbf{x}_n \leftarrow \mathbf{x}_n - \bar{\mathbf{x}}$
2. calculate $\tilde{d}$ top eigenvectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_{\tilde{d}}$ of $\mathrm{X}^T\mathrm{X}$
3. return feature transform $\Phi(\mathbf{x}) = \mathrm{W}(\mathbf{x} - \bar{\mathbf{x}})$

linear autoencoder与PCA也有差别，PCA是基于统计学分析得到的。一般我们认为，将高维数据投影（降维）到低维空间中，应该保证数据本身的方差越大越好，而噪声方差越小越好，而PCA正是基于此原理推导的。linear autoencoder与PCA都可以用来进行数据压缩，但是PCA应用更加广泛一些。

- linear autoencoder:
  maximize $\sum$(maginitude after projection)^2
- principal component analysis (PCA) from statistics:
  maximize $\sum$(variance after projection)
- both useful for linear dimension reduction
  though PCA more popular

以上关于PCA的推导基本上是从几何的角度，而没有从代数角度进行详细的数学推导。网上关于PCA的资料很多，这里附上一篇个人觉得讲解得通俗易懂的PCA原理介绍：PCA的数学原理。有兴趣的朋友可以看一看。

总结

本节课主要介绍了深度学习（deep learning）的数学模型，也是上节课讲的神经网络的延伸。由于深度学习网络的复杂性，其建模优化是比较困难的。通常，我们可以从pre-training和regularization的角度来解决这些困难。首先，autoencoder可以得到比较不错的初始化权重，起到pre-training的效果。然后，denoising autoencoder通过引入人工噪声，训练得到初始化权重，从而使模型本身抗噪声能力更强，更具有健壮性，起到了regularization的效果。最后，我们介绍了linear autoencoder并从几何角度详述了其推导过程。linear autoencoder与PCA十分类似，都可以用来进行数据压缩和数据降维处理。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程



林轩田《机器学习技法》课程笔记14 -- Radial Basis Function Network

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了Deep Learning的概念。Deep Learing其实是Neural Networ的延伸，神经元更多，网络结构更加复杂。深度学习网络在训练的过程中最核心的问题就是pre-training和regularization。pre-training中，我们使用denoising autoencoder来对初始化权重进行选择。denoising autoencoder与统计学中经常用来进行数据处理的PCA算法具有很大的关联性。这节课我们将介绍Radial Basis Function Network，把之前介绍的adial Basis Function和Neural Network联系起来。

RBF Network Hypothesis

之前我们介绍过，在SVM中引入Gaussian Kernel就能在无限多维的特征转换中得到一条“粗壮”的分界线（或者高维分界平面、分界超平面）。从结果来看，Gaussian SVM其实就是将一些Gaussian函数进行线性组合，而Gaussian函数的中心就位于Support Vectors上，最终得到预测模型$g_{svm}(x)$。

$$g_{SVM}(\mathbf{x}) = \text{sign}\left(\sum_{SV} \alpha_n y_n \exp\left(-\gamma\|\mathbf{x}-\mathbf{x}_n\|^2\right) + b\right)$$

Gaussian SVM: find α_n to combine Gaussians centered at $\mathbf{x}_n$;
achieve large margin in infinite-dimensional space, remember? :-)

Gaussian kernel的另一种叫法是Radial Basis Function(RBF) kernel，即径向基函数。这个名字从何而来？首先，radial表示Gaussian函数计算结果只跟新的点x与中心点x_n的距离有关，与其它无关。basis function就是指Gaussian函数，最终的矩$g_{svm}(x)$就是由这些basis function线性组合而成。

从另外一个角度来看Gaussian SVM。首先，构造一个函数$g_n(x)$：

$$g_n(x) = y_n e^{-\gamma\|x-x_n\|^2}$$



上式中，指数项表示新的点x与x_n之间的距离大小。距离越近，即权重越大，相当于对y_n投的票数更多；而距离越远，权重越小，相当于对y_n投的票数更少。其物理意义是新的点与x_n的距离远近决定了$g_n(x)$与y_n的接近程度。如果距离越近，则y_n对$g_n(x)$的权重影响越大；如果距离越远，则y_n对$g_n(x)$的权重影响越小。那么整体来说，$g_{svm}(x)$就由所有的SV组成的$g_n(x)$线性组合而成，不同$g_n(x)$对应的系数是α_n，最后由sign函数做最后的选择。这个过程很类型我们之前介绍的aggregation中将所有较好的hypothesis线性组合，不同的$g_n(x)$有不同的权重α_n。我们把$g_n(x)$叫做radial hypotheses，Gaussian SVM就是将所有SV对应的radial hypotheses进行线性组合（linear aggregation）。

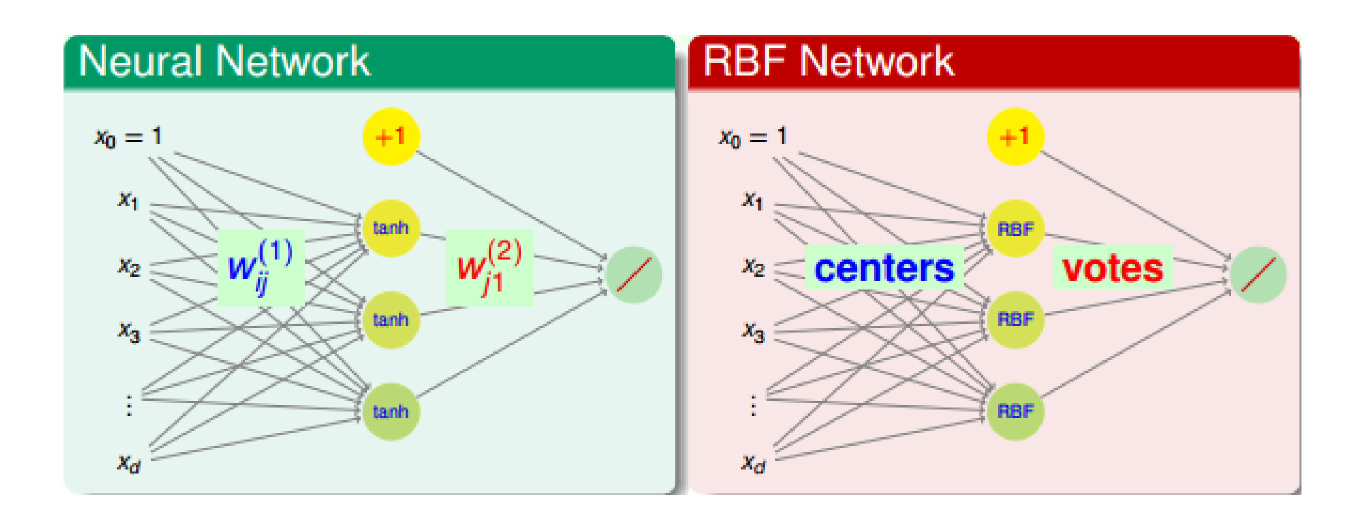
- Gaussian kernel: also called Radial Basis Function (RBF) kernel
  - radial: only depends on distance between $\mathbf{x}$ and 'center' $\mathbf{x}_n$
  - basis function: to be 'combined'
- let $g_n(\mathbf{x}) = y_n \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_n\|^2\right)$:

$$g_{\text{SVM}}(\mathbf{x}) = \text{sign}\left(\sum_{\text{SV}} \alpha_n g_n(\mathbf{x}) + b\right)$$

—linear aggregation of selected radial hypotheses

那么，Radial Basis Function(RBF) Network其实就是上面Gaussian SVM概念的延伸，目的就是找到所有radial hypotheses的linear aggregation，得到更好的网络模型。

之所以叫作RBF Network是因为它的模型结构类似于我们之前介绍的Neural Network。



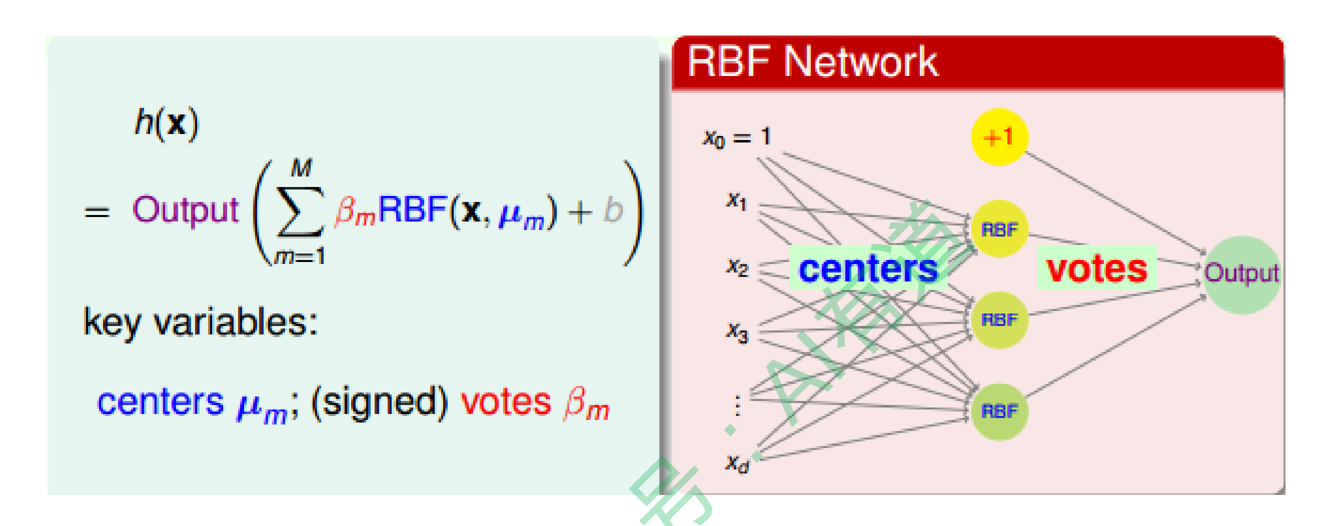
Neural Network与RBF Network在输出层基本是类似的，都是上一层hypotheses的线性组合（linear aggregation）。但是对于隐藏层的各个神经元来说，Neural Network是使用内积（inner-product）加上tanh()函数的方法，而RBF Network是使用距离



（distance）加上Gaussian函数的方法。总的来说，RBF Network是Neural Network的一个分支。

- hidden layer different:
  (inner-product + tanh) versus (distance + Gaussian)
- output layer same: **just linear aggregation**

至此，RBF Network Hypothesis以及网络结构可以写成如下形式：



上式中，μ_m表示每个中心点的位置，隐藏层每个神经元对应一个中心点；β_m表示每个RBF的权重，即投票所占比重。

对应到Gaussian SVM上，上式中的RBF就是Gaussian函数。由于是分类问题，上式中的Output就是sign函数。其中，RBF的个数M就等于支持向量的个数SV，μ_m就代表每个SV的坐标x_m，而β_m就是在Dual SVM中推导得到的$\alpha_n y_m$值。那我们学习的目标就是根据已知的RBF和Output，来决定最好的中心点位置μ_m和权重系数β_m。

g_{SVM} for Gaussian-SVM

- RBF: Gaussian; Output: sign (binary classification)
- $M = \#\text{SV}$; μ_m: SVM SVs $\mathbf{x}_m$; β_m: $\alpha_m y_m$ from SVM Dual

learning: given RBF and Output,
decide μ_m and β_m

在之前介绍SVM的时候，我们就讲过Mercer定理：一个矩阵是Kernel的充分必要条件是它是对称的且是半正定的，条件比较苛刻。除了Gaussian kernel还有Polynomial



kernel等等。Kernel实际上描述了两个向量之间的相似性，通过转换到z空间计算内积的方式，来表征二者之间的相似性。而RBF实际上是直接使用x空间的距离来描述了一种相似性，距离越近，相似性越高。因此，kernel和RBF可以看成是两种衡量相似性（similarity）的方式。本文介绍的Gaussian RBF即为二者的交集。

kernel: similarity via $\mathcal{Z}$-space inner product
—governed by Mercer's condition, remember? :-)
$\text{Poly}(\mathbf{x}, \mathbf{x}') = (1 + \mathbf{x}^T\mathbf{x}')^2$

$\text{Gaussian}(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$

$\text{Truncated}(\mathbf{x}, \mathbf{x}') = [\![\|\mathbf{x} - \mathbf{x}'\| \le 1]\!]\,(1 - \|\mathbf{x} - \mathbf{x}'\|)^2$
RBF: similarity via $\mathcal{X}$-space distance
—often monotonically non-increasing to distance

除了kernel和RBF之外，还有其它衡量相似性的函数。例如神经网络中的神经元就是衡量输入和权重之间的相似性。

经过以上分析，我们知道了RBF Network中distance similarity是一个很好的定义特征转换的方法。除此之外，我们还可以使用其它相似性函数来表征特征转换，从而得到更好的机器学习模型。

RBF Network Learning

我们已经介绍了RBF Network的Hypothesis可表示为：

$$h(\mathbf{x}) = \text{Output}\left(\sum_{m=1}^{M} \beta_m \text{RBF}(\mathbf{x}, \boldsymbol{\mu}_m)\right)$$

其中μ_m表示中心点的位置。μ_m的个数M是人为决定的，如果将每个样本点x_m都作为一个中心点，即M=N，则我们把这种结构称为full RBF Network。也就是说，对于full RBF Network，每个样本点都对最终的预测都有影响（uniform influence），影响的程度由距离函数和权重β_m决定。如果每个样本点的影响力都是相同的，设为1，$\beta_m = 1 \cdot y_m$，那么相当于只根据距离的远近进行投票。最终将x与所有样本点的RBF距离线性组合，经过sign函数后，得到最终的预测分类结果。这实际上就是aggregation的过程，考虑并计入所有样本点的影响力，最后将x与所有样本点的distance similarity进行线性组合。



- full RBF Network: $M = N$ and each $\mu_m = \mathbf{x}_m$
- physical meaning: each $\mathbf{x}_m$ **influences** similar $\mathbf{x}$ by β_m
- e.g. uniform influence with $\beta_m = 1 \cdot y_m$ for binary classification

$$g_{\text{uniform}}(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{N} y_m \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}_m\|^2\right)\right)$$

—**aggregate** each example's opinion subject to similarity

full RBF Network的矩可以表示为：

$$g_{\text{uniform}}(\mathbf{x}) = \text{sign}\left(\sum_{m=1}^{N} y_m \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}_m\|^2\right)\right)$$

我们来看上式中的Gaussian函数项，当x与样本点x_m越接近的时候，其高斯函数值越大。由于Gaussian函数曲线性质，越靠近中心点，值越大；偏离中心点，其值会下降得很快。也就是说，在所有N个中心样本点中，往往只有距离x最近的那个样本点起到关键作用，而其它距离x较远的样本点其值很小，基本可以忽略。因此，为了简化运算，我们可以找到距离x最近的中心样本点，只用这一个点来代替所有N个点，最后得到的矩$g_{nbor}(x)$也只由该最近的中心点决定。这种模型叫做nearest neighbor model，只考虑距离x最近的那一个“邻居”。

当然可以对nearest neighbor model进行扩展，如果不是只选择一个“邻居”，而是选择距离x最近的k个“邻居”，进行uniformly aggregation，得到最终的矩$g_{nbor}(x)$。这种方法通常叫做k近邻算法（k nearest neighbor）。

- $\exp\left(-\gamma \|\mathbf{x} - \mathbf{x}_m\|^2\right)$: maximum when $\mathbf{x}$ **closest to** $\mathbf{x}_m$
  —maximum one often dominates the $\sum_{m=1}^{N}$ term
- take y_m of **maximum** $\exp(\ldots)$ instead of voting of all y_m
  —**selection** instead of **aggregation**
- physical meaning:

$$g_{\text{nbor}}(\mathbf{x}) = y_m \text{ such that } \mathbf{x} \text{ closest to } \mathbf{x}_m$$

  —called **nearest neighbor** model
- can **uniformly aggregate** k neighbors also: k **nearest neighbor**



k nearest neighbor通常比nearest neighbor model效果更好，计算量上也比full RBF Network要简单一些。值得一提的是，k nearest neighbor与full RBF Network都是比较“偷懒”的方法。因为它们在训练模型的时候比较简单，没有太多的运算，但是在测试的时候却要花费更多的力气，找出最相近的中心点，计算相对复杂一些。

接下来，我们来看一下Full RBF Network有什么样的优点和好处。考虑一个squared error regression问题，且每个RBF的权重为β_m而不是前面简化的y_m。目的是计算最优化模型对应的β_m值。该hypothesis可表示为：

full RBF Network for squared error regression:

$$h(\mathbf{x}) = \text{~~Output~~}\left(\sum_{m=1}^{N} \beta_m \text{RBF}(\mathbf{x}, \mathbf{x}_m)\right)$$

很明显，这是一个简单的线性回归问题，每个RBF都可以看成是特征转换。特征转换后的向量z_n可表示为：

$$z_n = [RBF(x_n, x_1),\ RBF(x_n, x_2),\ \cdots,\ RBF(x_n, x_N)]$$

那么，根据之前线性回归介绍过的最优化解公式，就能快速地得到β的最优解为：

$$\beta = (Z^T Z)^{-1} Z^T y$$

上述解的条件是矩阵$Z^T Z$是可逆的。

矩阵Z的大小是NxN，是一个方阵。而且，由于Z中每个向量z_n表示该点与其它所有点的RBF distance，所以从形式上来说，Z也是对称矩阵。如果所有的样本点x_n都不一样，则Z一定是可逆的。

- just linear regression on RBF-transformed data

$$\mathbf{z}_n = [\text{RBF}(\mathbf{x}_n, \mathbf{x}_1), \text{RBF}(\mathbf{x}_n, \mathbf{x}_2), \ldots, \text{RBF}(\mathbf{x}_n, \mathbf{x}_N)]$$

- optimal β? $\beta = (\mathrm{Z}^T\mathrm{Z})^{-1}\mathrm{Z}^T\mathbf{y}$, if $\mathrm{Z}^T\mathrm{Z}$ invertible, remember? :-)
- size of Z? N (examples) by N (centers)
  —symmetric square matrix
- theoretical fact: if $\mathbf{x}_n$ all different, Z with Gaussian RBF invertible

根据Z矩阵的这些性质，我们可以对β的解进行化简，得到：



$$\beta = Z^{-1}y$$

将β的解代入矩的计算中，以x_1为例，得到：

$$g_{RBF}(x_1) = \beta^T z_1 = y^T Z^{-1} z_1 = y^T \ [1\ 0\ \cdots\ 0]^T = y_1$$

结果非常有趣，模型的输出与原样本y_1完全相同。同样，对任意的x_n，都能得到$g_{RBF}(x_n) = y_n$。因此，$E_{in}(g_{RBF}) = 0$。看起来，这个模型非常完美了，没有error。但是，我们之前就说过，机器学习中，$E_{in} = 0$并非好事，很可能造成模型复杂度增加及过拟合。

full Gaussian RBF Network for regression: $\beta = Z^{-1}\mathbf{y}$

$g_{RBF}(\mathbf{x}_1) = \beta^T \mathbf{z}_1 = \mathbf{y}^T Z^{-1}(\text{first column of } Z) = \mathbf{y}^T [\ 1\ \ 0\ \ \ldots\ \ 0\ ]^T = y_1$

—$g_{RBF}(\mathbf{x}_n) = y_n$, i.e. $E_{in}(g_{RBF}) = 0$, yeah!! :-)

当然，这种方法在某些领域还是很有用的。比如在函数拟合（function approximation）中，目标就是让$E_{in} = 0$，使得原所有样本都尽可能地落在拟合的函数曲线上。

为了避免发生过拟合，我们可以引入正则项λ，得到β的最优解为：

$$\beta = (Z^T Z + \lambda I)^{-1} Z^T y$$

- called exact interpolation for function approximation
- but overfitting for learning? :-(
- how about regularization? e.g. ridge regression for β instead
  —optimal $\beta = (Z^T Z + \lambda I)^{-1} Z^T \mathbf{y}$
- seen Z? $Z = [\text{Gaussian}(\mathbf{x}_n, \mathbf{x}_m)]$ = Gaussian kernel matrix K

我们再来看一下Z矩阵，Z矩阵是由一系列Gaussian函数组成，每个Gaussian函数计算的是两个样本之间的distance similarity。这里的Z与之前我们介绍的Gaussian SVM中的kernel K是一致的。当时我们得到kernel ridgeregression中线性系数β的解为：

$$\beta = (K + \lambda I)^{-1} y$$

比较一下kernel ridgeregression与regularized full RBF Network的β解，形式上相似但不完全相同。这是因为regularization不一样，在kernel ridgeregression中，是对无限



多维的特征转换做regularization，而在regularized full RBF Network中，是对有限维（N维度）的特征转换做regularization。因此，两者的公式解有细微差别。

effect of regularization in different spaces:

kernel ridge regression: $\beta = (\mathrm{K} + \lambda \mathrm{I})^{-1}\mathbf{y}$;
regularized full RBFNet: $\beta = (\mathrm{Z}^T\mathrm{Z} + \lambda \mathrm{I})^{-1}\mathrm{Z}^T\mathbf{y}$

除此之外，还有另外一种regularization的方法，就是不把所有N个样本点都拿来作中心点，而是只选择其中的M个样本点作为中心点。类似于SVM中的SV一样，只选择具有代表性的M个中心点。这样减少中心点数量的同时也就减少了权重的数量，能够起到regularization的效果，避免发生过拟合。

recall:

$$g_{\mathrm{SVM}}(\mathbf{x}) = \mathrm{sign}\left(\sum_{\mathrm{SV}} \alpha_m y_m \exp\left(-\gamma\|\mathbf{x} - \mathbf{x}_m\|^2\right) + b\right)$$

—only '$\ll N$' SVs needed in 'network'

- next: $M \ll N$ instead of $M = N$
- effect: **regularization** by constraining **number of centers and voting weights**
- physical meaning of centers μ_m: **prototypes**

下一部分，我们将讨论如何选取M个中心点作为好的代表。

k-Means Algorithm

之所以要选择代表，是因为如果某些样本点很接近，那么就可以用一个中心点来代表它们。这就是聚类（cluster）的思想，从所有N个样本点中选择少数几个代表作为中心点。

if $\mathbf{x}_1 \approx \mathbf{x}_2$,
$\Longrightarrow$ **no need** both $\mathrm{RBF}(\mathbf{x}, \mathbf{x}_1)$ & $\mathrm{RBF}(\mathbf{x}, \mathbf{x}_2)$ in RBFNet,
$\Longrightarrow$ **cluster** $\mathbf{x}_1$ and $\mathbf{x}_2$ by **one prototype** $\mu \approx \mathbf{x}_1 \approx \mathbf{x}_2$



聚类（clustering）问题是一种典型的非监督式学习（unsupervised learning）。它的优化问题有两个变量需要确定：一个是分类的分群值S_m，每一类可表示为$S_1, S_2, \cdots, S_M$；另外一个是每一类对应的中心点$\mu_1, \mu_2, \cdots, \mu_M$。那么对于该聚类问题的优化，其error function可使用squared error measure来衡量。

- clustering with prototype:
  - partition $\{\mathbf{x}_n\}$ to disjoint sets $S_1, S_2, \cdots, S_M$
  - choose μ_m for each S_m

  —hope: $\mathbf{x}_1, \mathbf{x}_2$ both $\in S_m \Leftrightarrow \mu_m \approx \mathbf{x}_1 \approx x_2$
- cluster error with squared error measure:

$$E_{\text{in}}(S_1, \cdots, S_M; \mu_1, \cdots, \mu_M) = \frac{1}{N} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \mu_m\|^2$$

那么，我们的目标就是通过选择最合适的$S_1, S_2, \cdots, S_M$和$\mu_1, \mu_2, \cdots, \mu_M$，使得$E_{in}$最小化。对应的公式可表示为：

with $S_1, \cdots, S_M$ being a partition of $\{\mathbf{x}_n\}$,

$$\min_{\{S_1, \cdots, S_M; \mu_1, \cdots, \mu_M\}} \sum_{n=1}^{N} \sum_{m=1}^{M} [\![\mathbf{x}_n \in S_m]\!] \|\mathbf{x}_n - \mu_m\|^2$$

- hard to optimize: joint combinatorial-numerical optimization
- two sets of variables: will optimize alternatingly

从这个最小化公式，我们能够发现这是一个组合最佳化的问题，既要优化分群值S_m，又要求解每一类的中心点u_m。所以，这个最小化问题是比较复杂、难优化的。通常的办法是对S和μ分别进行最优化求解。

首先，如果$\mu_1, \mu_2, \cdots, \mu_M$是固定的，目标就是只要对所有的$x_n$进行分群归类。这个求解过程很简单，因为每个样本点只能属于一个群S，不能同时属于两个或多个群。所以，只要根据距离公式，计算选择离x_n最近的中心点μ即可。



if $\mu_1, \cdots, \mu_M$ **fixed**, for each $\mathbf{x}_n$

- $[\![\mathbf{x}_n \in S_m]\!]$: choose one and only one subset
- $\|\mathbf{x}_n - \mu_m\|^2$: distance to each prototype

optimal chosen subset $S_m =$ the one with **minimum** $\|\mathbf{x}_n - \mu_m\|^2$

for given $\mu_1, \cdots, \mu_M$, each $\mathbf{x}_n$
'optimally partitioned' using its **closest** μ_m

然后，如果$S_1, S_2, \cdots, S_M$是固定的，目标就是只要找出每个类的中心点μ。显然，根据上式中的error function，所有的x_n分群是已知的，那么该最小化问题就是一个典型的数值最优化问题。对于每个类群S_m，利用梯度下降算法，即可得到μ_m的解。

if $S_1, \cdots, S_M$ **fixed**, just **unconstrained optimization** for each μ_m

$$\nabla_{\mu_m} E_{\text{in}} = -2 \sum_{n=1}^{N} [\![\mathbf{x}_n \in S_m]\!](\mathbf{x}_n - \mu_m) = -2\left(\left(\sum_{\mathbf{x}_n \in S_m} \mathbf{x}_n\right) - |S_m|\mu_m\right)$$

optimal prototype $\mu_m =$ **average** of $\mathbf{x}_n$ within S_m

for given $S_1, \cdots, S_M$, each μ_n
'optimally computed' as **consensus** within S_m

如上图所示，中心点μ_m就等于所有属于类群S_m的平均位置处。

经过以上的推导，我们得到了一个非常有名的一种unsupervised learning算法，叫做k-Means Algorithm。这里的k就是代表上面的M，表示类群的个数。

k-Means Algorithm的流程是这样的：首先，随机选择k个中心点$\mu_1, \mu_2, \cdots, \mu_k$；然后，再由确定的中心点得到不同的类群$S_1, S_2, \cdots, S_k$；接着，再由确定的类群计算出新的不同的k个中心点；继续循环迭代计算，交互地对μ和S值进行最优化计算，不断更新μ和S值，直到程序收敛，实现E_{in}最小化。具体算法流程图如下所示：



k-Means Algorithm

1. initialize $\mu_1, \mu_2, \ldots, \mu_k$: say, as k randomly chosen $\mathbf{x}_n$
2. alternating optimization of E_{in}: repeatedly
   1. optimize $S_1, S_2, \ldots, S_k$:
      each $\mathbf{x}_n$ 'optimally partitioned' using its closest μ_i
   2. optimize $\mu_1, \mu_2, \ldots, \mu_k$:
      each μ_n 'optimally computed' as consensus within S_m

until converge

有一个问题是，k-Means Algorithm的循环迭代一定会停止吗？或者说一定能得到最优解吗？答案是肯定的。因为每次迭代更新，μ和S值都会比上一次的值更接近最优解，也就是说E_{in}是不断减小的。而E_{in}的下界是0，所以，E_{in}最终会等于0，μ和S最终能得到最优解。

k-Means Algorithm已经介绍完毕。接下来，我们把k-Means Algorithm应用到RBF Network中去。首先，使用k-Means，得到原始样本的k个中心点。原始样本到k个中心点组成了RBF特征转换$\Phi(x)$。然后，根据上面介绍过的线性模型，由最优化公式解计算得到权重β值。最后，将所有的$\Phi(x)$用β线性组合，即得到矩$g_{RBFNET}(x)$的表达式。具体的算法流程如下所示：

RBF Network Using k-Means

1. run k-Means with $k = M$ to get $\{\mu_m\}$
2. construct transform $\Phi(\mathbf{x})$ from RBF (say, Gaussian) at μ_m

$$\Phi(\mathbf{x}) = [\text{RBF}(\mathbf{x}, \mu_1), \text{RBF}(\mathbf{x}, \mu_2), \ldots, \text{RBF}(\mathbf{x}, \mu_M)]$$

3. run linear model on $\{(\Phi(\mathbf{x}_n), y_n)\}$ to get β
4. return $g_{\text{RBFNET}}(\mathbf{x}) = \text{LinearHypothesis}(\beta, \Phi(\mathbf{x}))$

值得一提的是，这里我们使用了unsupervised learning（k-Means）与我们上节课介绍的autoencoder类似，同样都是特征转换（feature transform）的方法。

在最优化求解过程中，参数有k-Means类群个数M、Gaussian函数参数λ等。我们可以采用validation的方法来选取最佳的参数值。



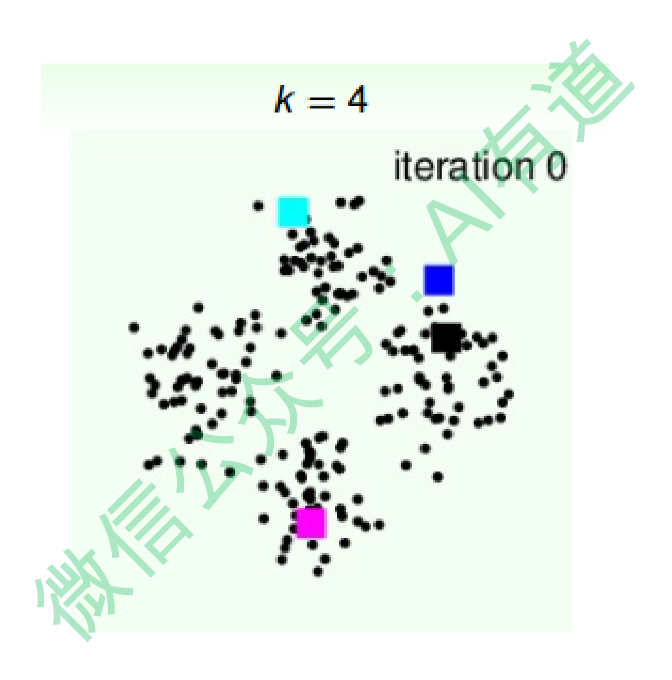
- using unsupervised learning (k-Means) to assist **feature transform**—like **autoencoder**
- parameters: M (prototypes), RBF (such as γ of Gaussian)

RBF Network: a simple (**old-fashioned**) model

k-means and RBF Network in Action

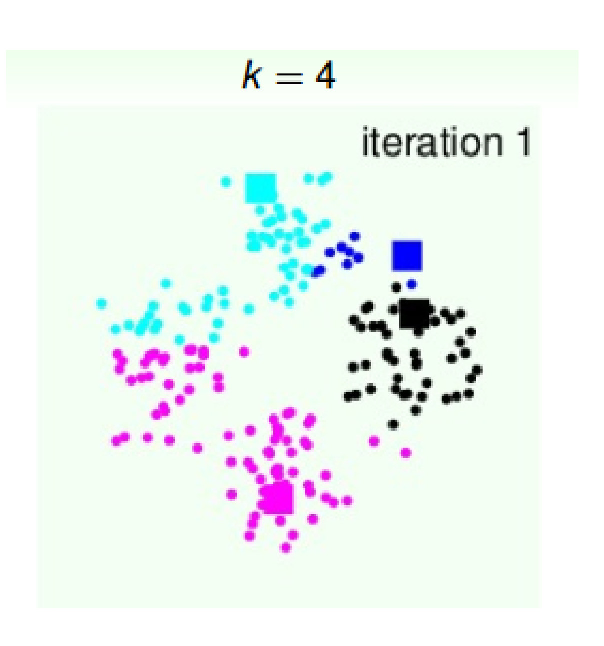
下面这部分，我们将举几个例子，看一下k-Means Algorithm是如何处理分类问题的。

第一个例子，平面上有4个类群，k=4。首先，我们随机选择4个中心点，如下图中四种颜色的方块所示：

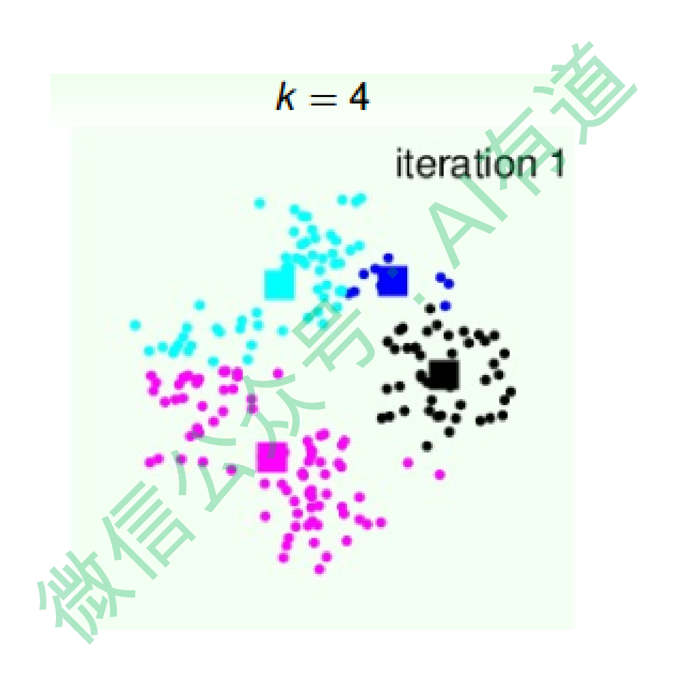


第一次迭代，由初始中心点，得到4个类群点的分布：



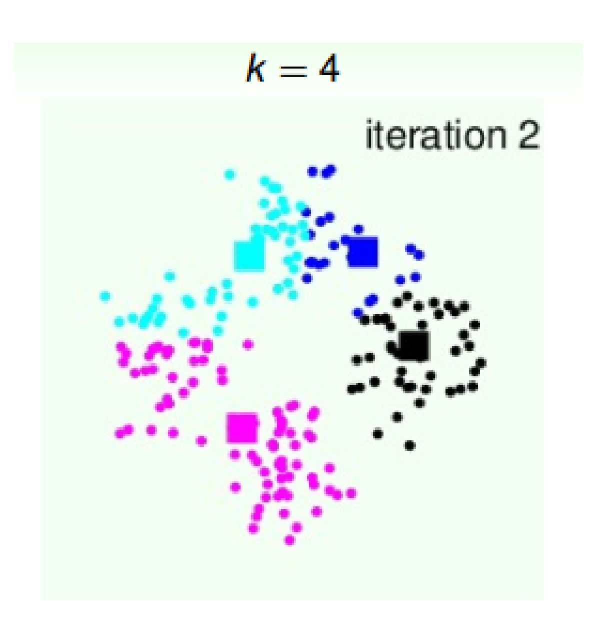


4个类群点确定后，再更新4个中心点的位置：

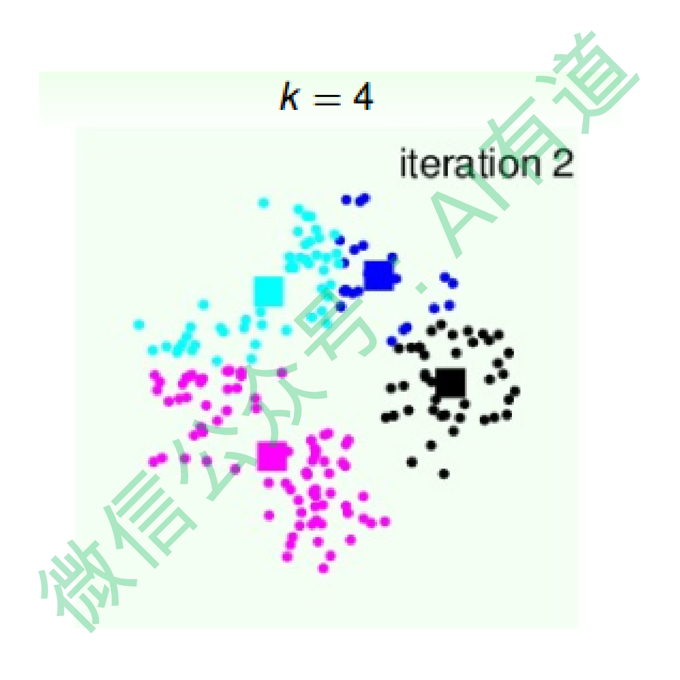


第二次迭代，由上面得到的4个中心点，再计算4个类群点的分布：



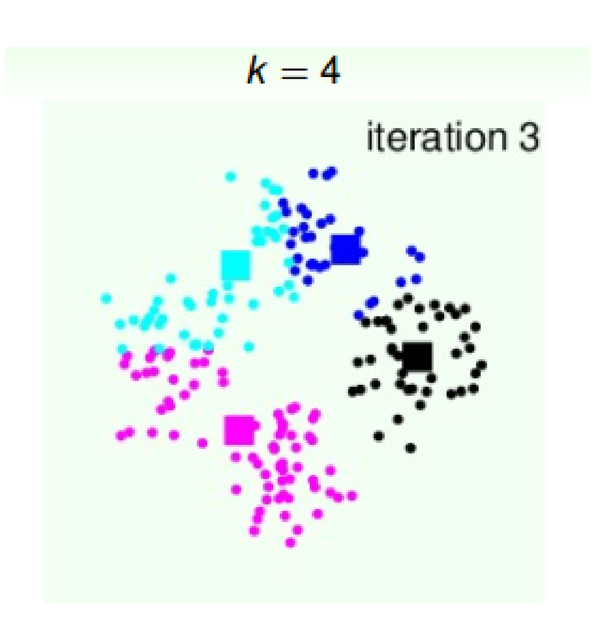


4个类群点确定后，再更新4个中心点的位置：

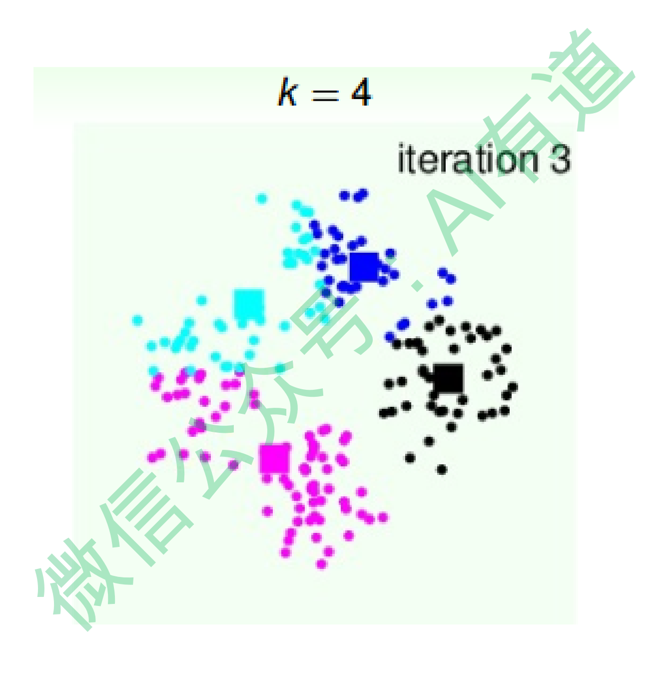


第三次迭代，由上面得到的4个中心点，再计算4个类群点的分布：



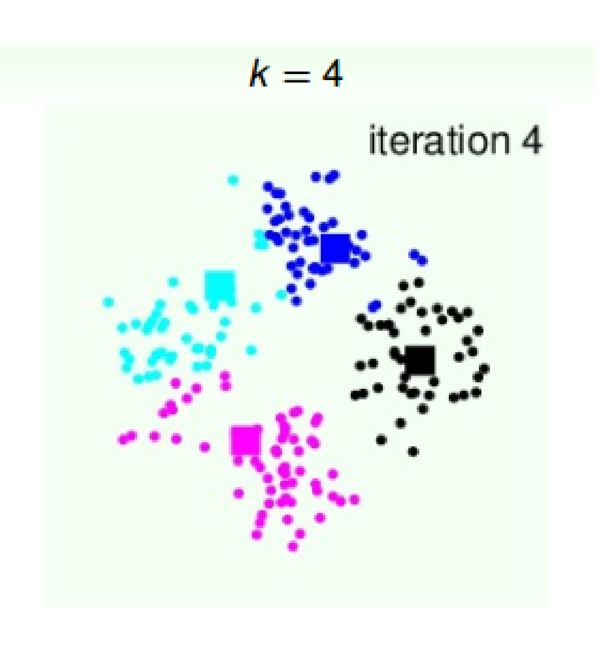


4个类群点确定后，再更新4个中心点的位置：

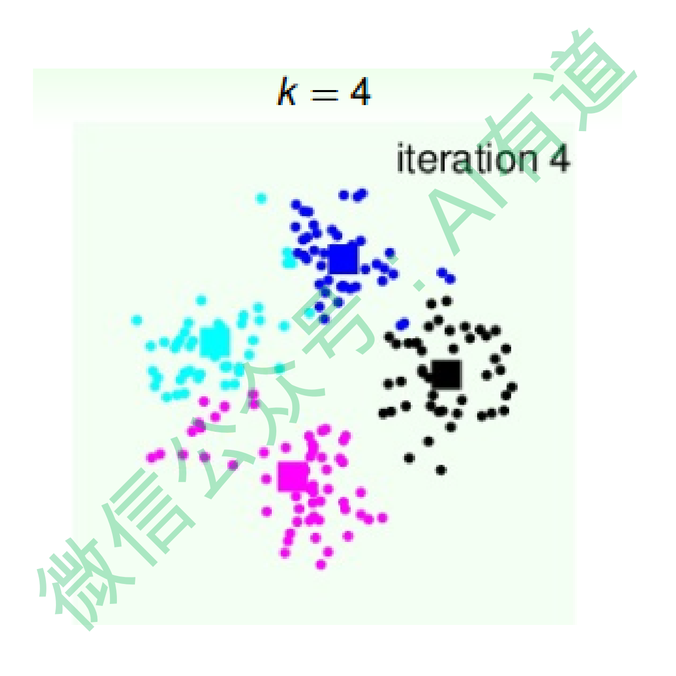


第四次迭代，由上面得到的4个中心点，再计算4个类群点的分布：



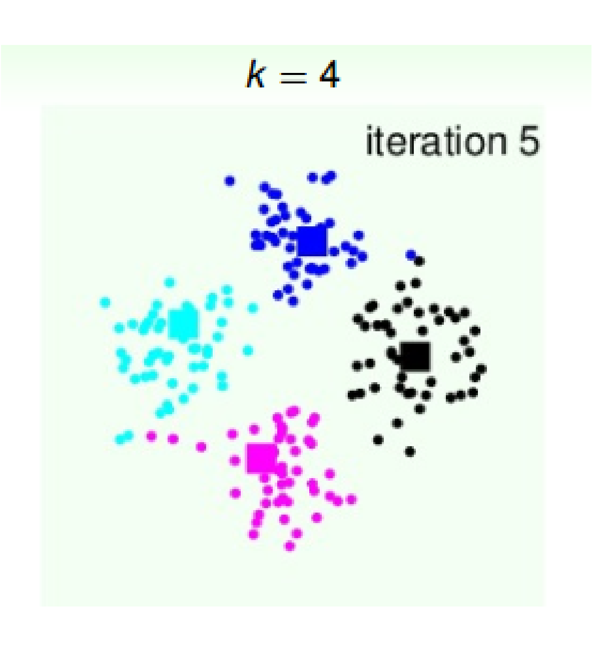


4个类群点确定后，再更新4个中心点的位置：

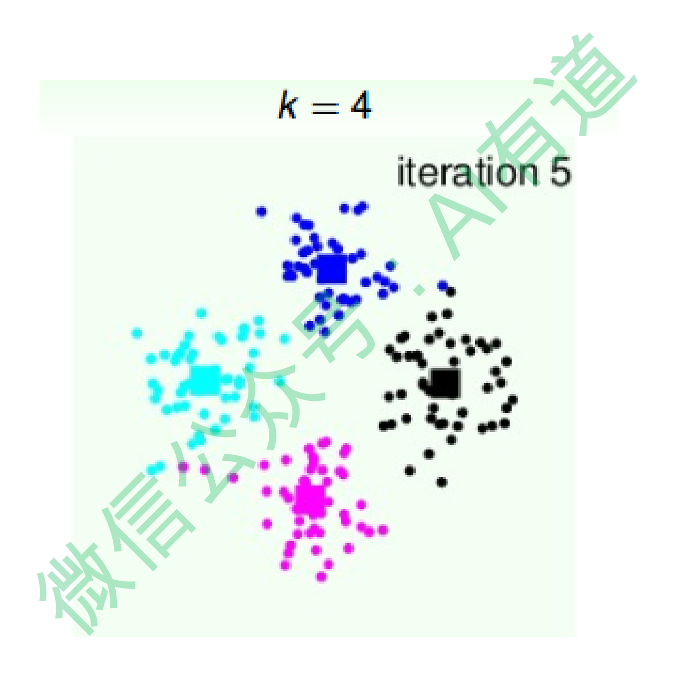


第五次迭代，由上面得到的4个中心点，再计算4个类群点的分布：



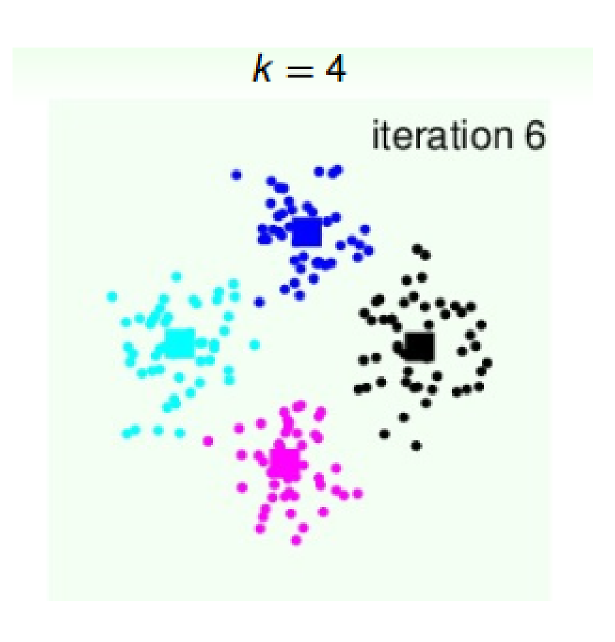


4个类群点确定后，再更新4个中心点的位置：

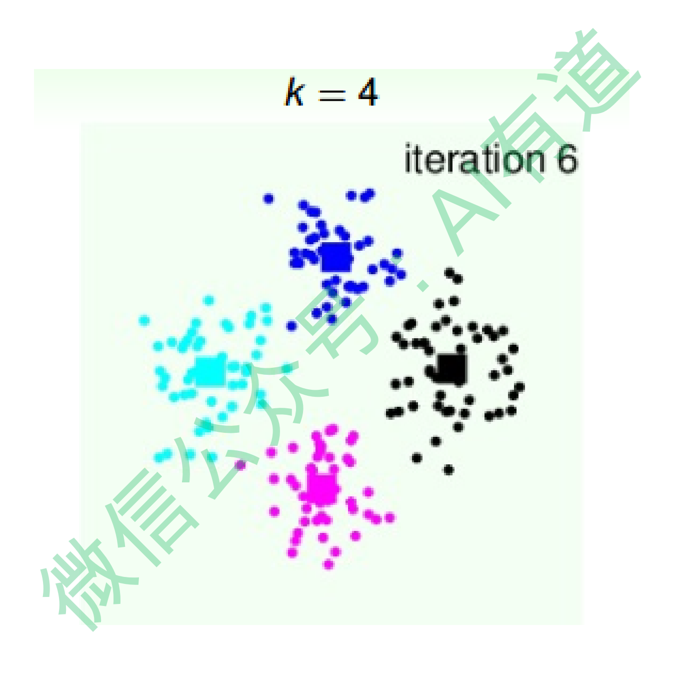


第六次迭代，由上面得到的4个中心点，再计算4个类群点的分布：



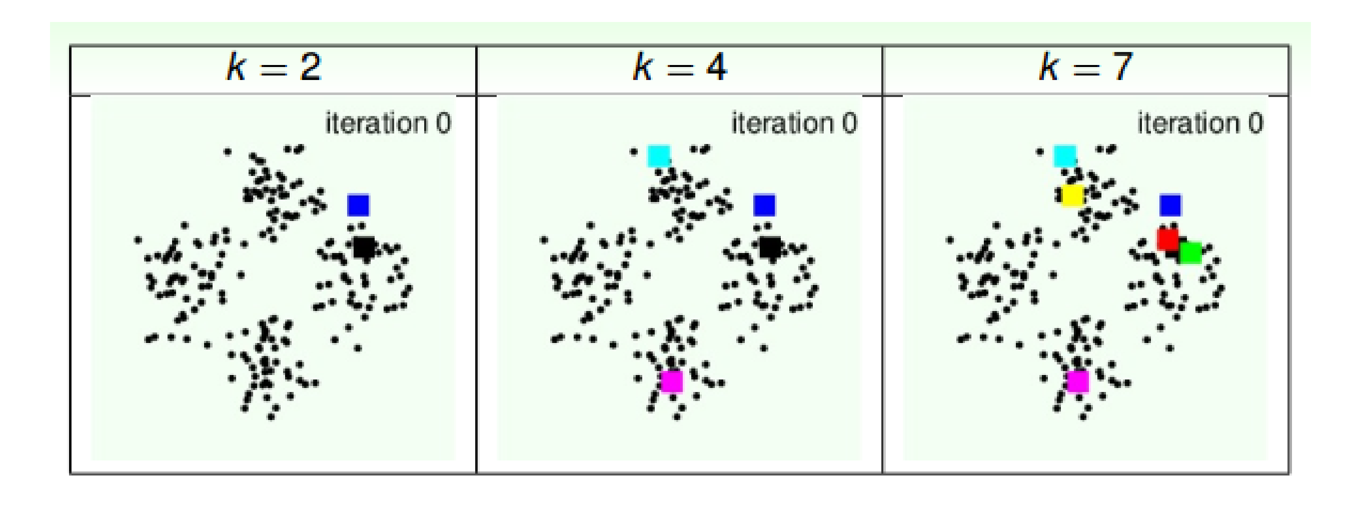


4个类群点确定后，再更新4个中心点的位置：

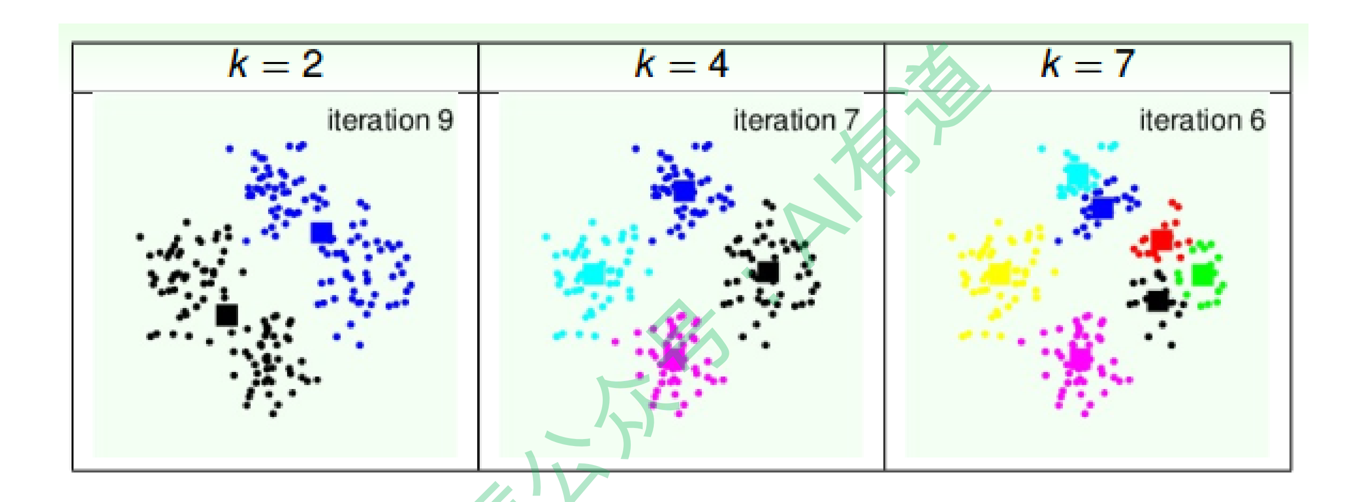


从上图我们可以看到，经过六次迭代计算后，聚类的效果已经相当不错了。从另外一个角度来说，k值的选择很重要，下面我们来看看不同的k值对应什么样的分类效果。





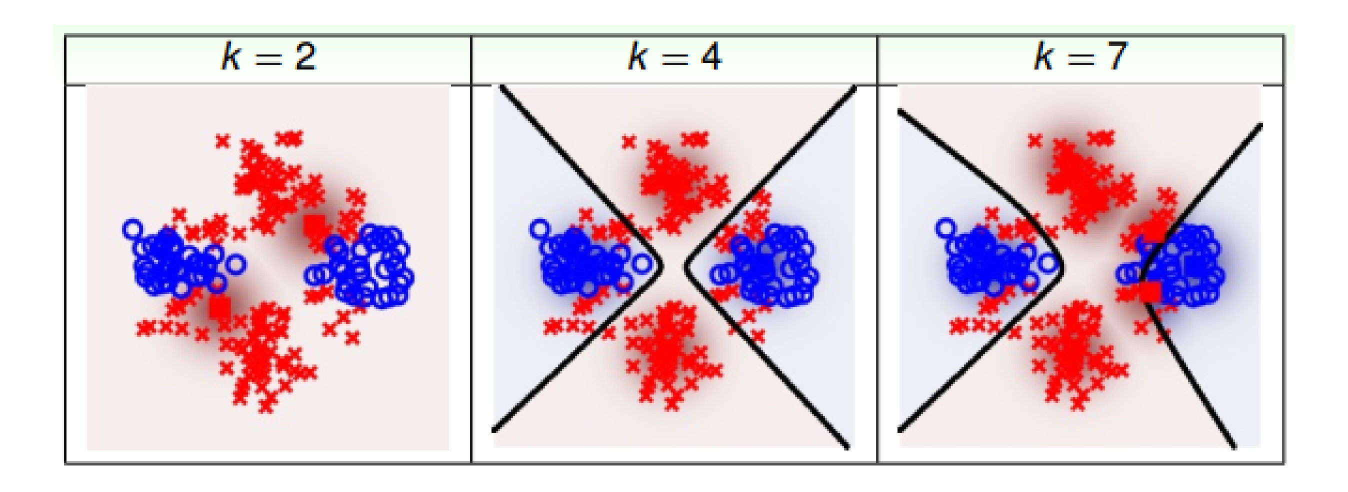
如上图所示，初始时，我们分别设定k为2，4，7，随机选择中心点位置。在经过多次迭代后，得到的聚类结果如下：



通过上面这个例子可以得出，不同的k值会得到不同的聚类效果。还有一点值得注意的是，初始中心点位置也可能会影响最终的聚类。例如上图中k=7的例子，初始值选取的右边三个中心点比较靠近，最后得到的右边三个聚类中心点位置也跟初始位置比较相近。所以，k值大小和初始中心点位置都会影响聚类效果。

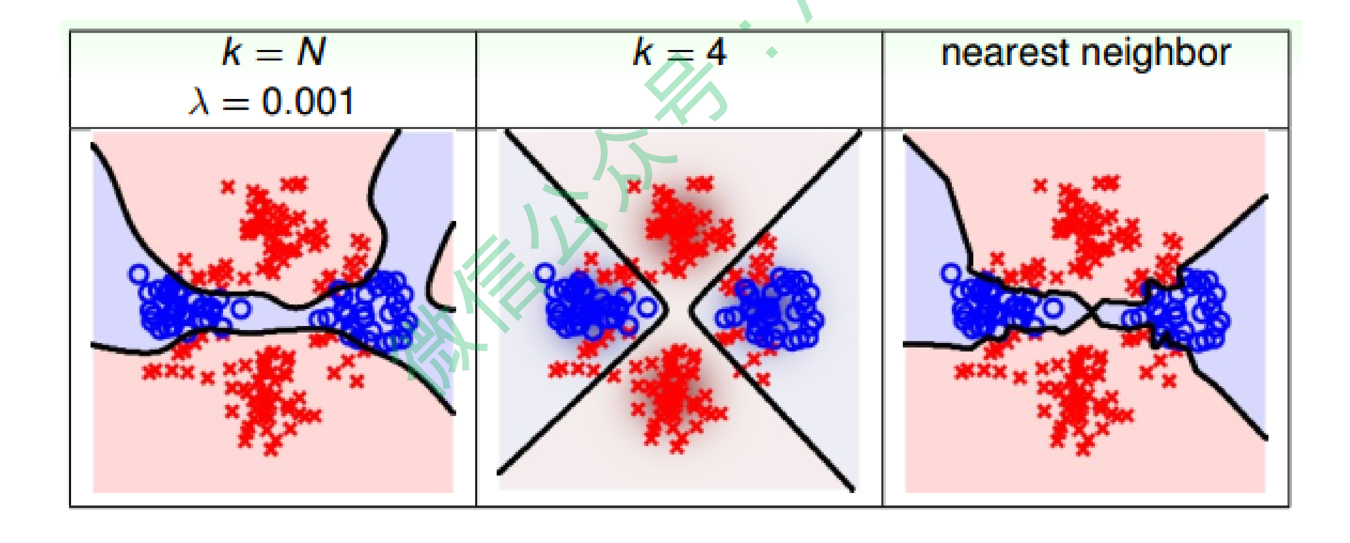
接下来，我们把k-Means应用到RBF Network中，同样分别设定k为2，4，7，不同模型得到的分类效果如下：





很明显，k=2时，分类效果不是太好；k=4时，分类效果好一些；而k=7时，分类效果更好，能够更细致地将样本准确分类。这说明了k-Means中k值设置得是否合理，对RBF Network的分类效果起到重要的作用。

再来看一个例子，如果使用full RBF Network进行分类，即k=N，如下图左边所示，设置正则化因子$\lambda = 0.001$。下图右边表示只考虑full RBF Network中的nearest neighbor。下图中间表示的是k=4的RBF Network的分类效果。



从上图的比较中，我们可以发现full RBF Network得到的分类线比较弯曲复杂。由于full RBF Network的计算量比较大，所以一般情况下，实际应用得不太多。

总结

本节课主要介绍了Radial Basis Function Network。RBF Network Hypothesis就是计算样本之间distance similarity的Gaussian函数，这类原型替代了神经网络中的神经元。RBF Network的训练学习过程，其实就是对所有的原型Hypotheses进行linear aggregation。然后，我们介绍了一个确定k个中心点的unsupervised learning算法，叫做k-Means Algorithm。这是一种典型的聚类算法，实现对原始样本数据的聚类分



群。接着，将k-Means Algorithm应用到RBF Network中，选择合适数量的中心点，得到更好的分类模型。最后，我们列举了几个在实际中使用k-Means和RBF Network的例子，结果显示不同的类群k值对分类的效果影响很大。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程





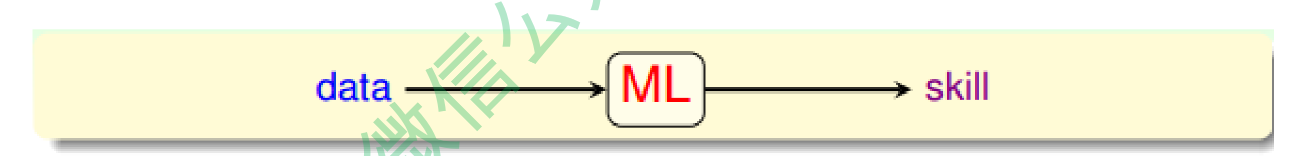
林轩田《机器学习技法》课程笔记15 -- Matrix Factorization

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了Radial Basis Function Network。它的原理就是基于距离相似性（distance-based similarities）的线性组合（linear aggregation）。我们使用k-Means clustering算法找出具有代表性的k个中心点，然后再计算与这些中心点的distance similarity，最后应用到RBF Network中去。

LinearNetwork Hypothesis

回顾一下，我们在机器学习基石课程的第一节课就提到过，机器学习的目的就是让机器从数据data中学习到某种能力skill。我们之前举过一个典型的推荐系统的例子。就是说，假如我们手上有许多不同用户对不同电影的排名rank，通过机器学习，训练一个模型，能够对用户没有看过的某部电影进行排名预测。



一个典型的电影推荐系统的例子是2006年Netflix举办的一次比赛。数据包含了480189个用户和17770部电影，总共1亿多个排名信息。该推荐系统模型中，我们用$\breve{x}_n = (n)$表示第n个用户，这是一个抽象的特征，常常使用数字编号来代替具体哪个用户。输出方面，我们使用$y_m = r_{nm}$表示第n个用户对第m部电影的排名数值。



A Hot Problem

- competition held by Netflix in 2006
  - 100,480,507 ratings that 480,189 users gave to 17,770 movies
  - 10% improvement = 1 million dollar prize
- data $\mathcal{D}_m$ for m-th movie:

$$\{(\tilde{\mathbf{x}}_n = (n), y_n = r_{nm}) : \text{ user } n \text{ rated movie } m\}$$

—abstract feature $\tilde{\mathbf{x}}_n = (n)$

下面我们来进一步看看这些抽象的特征，$\breve{x}_n = (n)$是用户的ID，通常用数字表示。例如1126,5566,6211等。这些编号并没有数值大小上的意义，只是一种ID标识而已。这类特征被称为类别特征（categorical features）。常见的categorical features包括：IDs，blood type，programming languages等等。而许多机器学习模型中使用的大部分都是数值特征（numerical features）。例如linear models，NNet模型等。但决策树（decision tree）是个例外，它可以使用categorical features。所以说，如果要建立一个类似推荐系统的机器学习模型，就要把用户ID这种categorical features转换为numerical features。这种特征转换其实就是训练模型之前一个编码（encoding）的过程。

- **categorical** features, e.g.
  - IDs
  - blood type: A, B, AB, O
  - programming languages: C, C++, Java, Python, ...
- many ML models operate on **numerical** features
  - **linear** models
  - **extended linear** models such as NNet

  —except for **decision trees**
- need: **encoding** (**transform**) from categorical to numerical

一种最简单的encoding方式就是binary vector encoding。也就是说，如果输入样本有N个，就构造一个维度为N的向量。第n个样本对应向量上第n个元素为1，其它元素都是0。下图就是一个binary vector encoding的例子。

binary vector encoding:

$$\text{A} = [1\ 0\ 0\ 0]^T, \text{B} = [0\ 1\ 0\ 0]^T,$$
$$\text{AB} = [0\ 0\ 1\ 0]^T, \text{O} = [0\ 0\ 0\ 1]^T$$



经过encoding之后，输入x_n是N维的binary vector，表示第n个用户。输出y_n是M维的向量，表示该用户对M部电影的排名数值大小。注意，用户不一定对所有M部电影都作过评价，未评价的恰恰是我们要预测的（下图中问号？表示未评价的电影）。

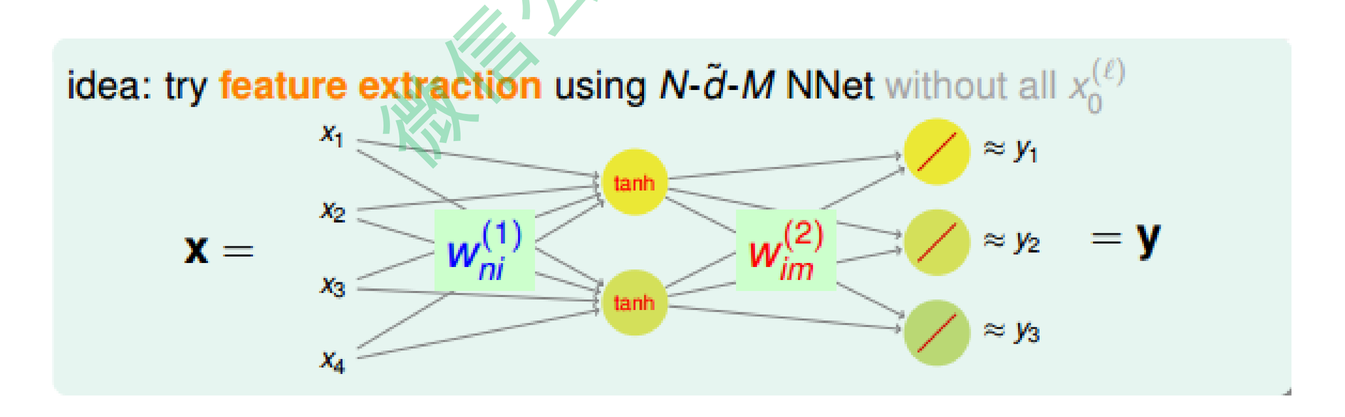
encoded data $\mathcal{D}_m$ for m-th movie:

$$\left\{\left(\mathbf{x}_n = \text{BinaryVectorEncoding}(n), y_n = r_{nm}\right): \text{ user } n \text{ rated movie } m\right\}$$

or, joint data $\mathcal{D}$

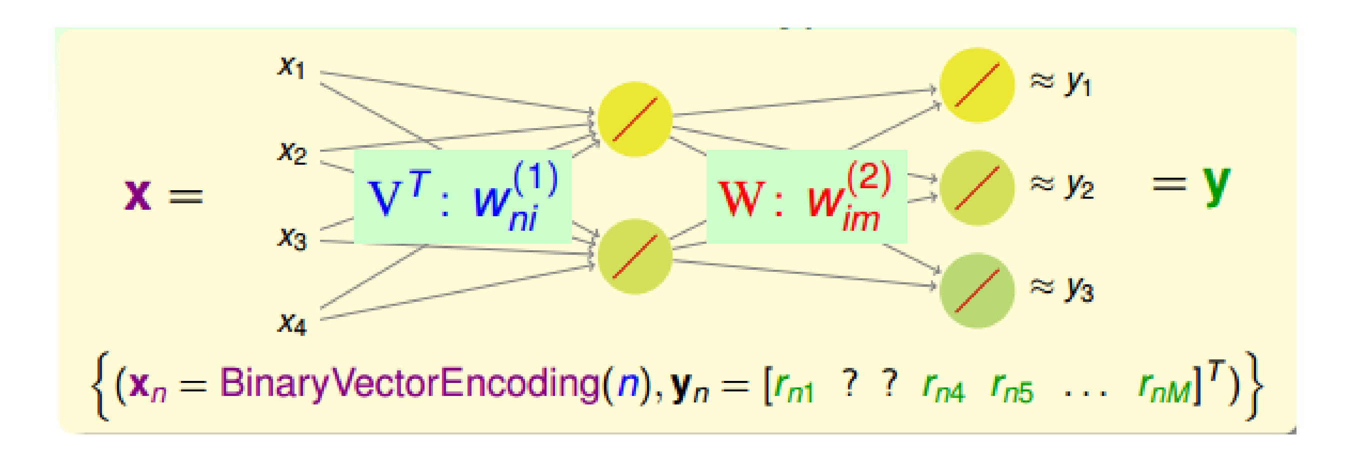
$$\left\{\left(\mathbf{x}_n = \text{BinaryVectorEncoding}(n), \mathbf{y}_n = [r_{n1} \;\; ? \;\; ? \;\; r_{n4} \;\; r_{n5} \;\; \ldots \;\; r_{nM}]^T\right)\right\}$$

总共有N个用户，M部电影。对于这样的数据，我们需要掌握每个用户对不同电影的喜爱程度及排名。这其实就是一个特征提取（feature extraction）的过程，提取出每个用户喜爱的电影风格及每部电影属于哪种风格，从而建立这样的推荐系统模型。可供选择使用的方法和模型很多，这里，我们使用的是NNet模型。NNet模型中的网络结构是$N-\breve{d}-M$型，其中N是输入层样本个数，$\breve{d}$是隐藏层神经元个数，M是输出层电影个数。该NNet为了简化计算，忽略了常数项。当然可以选择加上常数项，得到较复杂一些的模型。顺便提一下，这个结构跟我们之前介绍的autoencoder非常类似，都是只有一个隐藏层。



说到这里，有一个问题，就是上图NNet中隐藏层的tanh函数是否一定需要呢？答案是不需要。因为输入向量x是经过encoding得到的，其中大部分元素为0，只有一个元素为1。那么，只有一个元素x_n与相应权重的乘积进入到隐藏层。由于$x_n = 1$，则相当于只有一个权重值进入到tanh函数进行运算。从效果上来说，tanh(x)与x是无差别的，只是单纯经过一个函数的计算，并不影响最终的结果，修改权重值即可得到同样的效果。因此，我们把隐藏层的tanh函数替换成一个线性函数y=x，得到下图所示的结构。





由于中间隐藏层的转换函数是线性的，我们把这种结构称为Linear Network（与linear autoencoder比较相似）。看一下上图这个网络结构，输入层到隐藏层的权重$W_{ni}^{(1)}$维度是Nx$\breve{d}$，用向量V^T表示。隐藏层到输出层的权重$W_{im}^{(2)}$维度是$\breve{d}$xM，用矩阵W表示。把权重由矩阵表示之后，Linear Network的hypothesis 可表示为：

$$h(x) = W^T V x$$

如果是单个用户x_n，由于X向量中只有元素x_n为1，其它均为0，则对应矩阵V只有第n列向量是有效的，其输出hypothesis为：

$$h(x_n) = W^T v_n$$

- rename: V^T for $\left[w_{ni}^{(1)}\right]$ and W for $\left[w_{im}^{(2)}\right]$
- hypothesis: $\mathbf{h}(\mathbf{x}) = \mathrm{W}^T \mathrm{V} \mathbf{x}$
- per-user output: $\mathbf{h}(\mathbf{x}_n) = \mathrm{W}^T \mathbf{v}_n$, where $\mathbf{v}_n$ is n-th column of V

Basic Matrix Factorization

刚刚我们已经介绍了linear network的模型和hypothesis。其中Vx可以看作是对用户x的一种特征转换$\Phi(x)$。对于单部电影，其预测的排名可表示为：

$$h_m(x) = w_m^T \Phi(x)$$



linear network:

$$h(\mathbf{x}) = \mathrm{W}^T \underbrace{\mathrm{V}\mathbf{x}}_{\Phi(\mathbf{x})}$$

—for m-th movie, just linear model $h_m(\mathbf{x}) = \mathbf{w}_m^T \Phi(\mathbf{x})$
subject to shared transform Φ

推导完linear network模型之后，对于每组样本数据（即第n个用户第m部电影），我们希望预测的排名$w_m^T v_n$与实际样本排名y_n尽可能接近。所有样本综合起来，我们使用squared error measure的方式来定义E_{in}，E_{in}的表达式如下所示：

- for every $\mathcal{D}_m$, want $r_{nm} = y_n \approx \mathbf{w}_m^T \mathbf{v}_n$
- E_{in} over all $\mathcal{D}_m$ with squared error measure:

$$E_{\text{in}}(\{\mathbf{w}_m\}, \{\mathbf{v}_n\}) = \frac{1}{\sum_{m=1}^{M} |\mathcal{D}_m|} \sum_{\text{user } n \text{ rated movie } m} \left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right)^2$$

上式中，灰色的部分是常数，并不影响最小化求解，所以可以忽略。接下来，我们就要求出E_{in}最小化时对应的V和W解。

我们的目标是让真实排名与预测排名尽可能一致，即$r_{nm} \approx w_m^T v_n = v_n^T w_m$。把这种近似关系写成矩阵的形式：$R \approx V^T W$。矩阵R表示所有不同用户不同电影的排名情况，维度是NxM。这种用矩阵的方式进行处理的方法叫做Matrix Factorization。

$$r_{nm} \approx \mathbf{w}_m^T \mathbf{v}_n = \mathbf{v}_n^T \mathbf{w}_m \iff \mathrm{R} \approx \mathrm{V}^T \mathrm{W}$$

R	movie_1	movie_2	···	movie_M
user_1	100	80	···	?
user_2	?	70	···	90
···	···	···	···	···
user_N	?	60	···	0

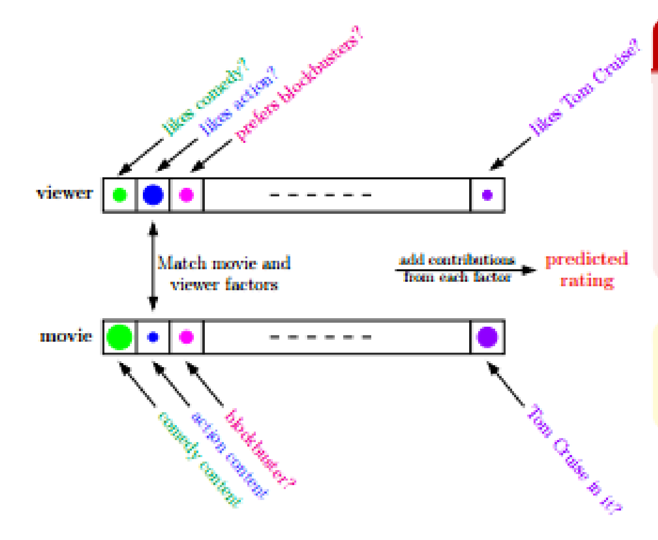
$\approx$

V^T
$-\mathbf{v}_1^T-$
$-\mathbf{v}_2^T-$
···
$-\mathbf{v}_N^T-$

W	$\mathbf{w}_1$	$\mathbf{w}_2$	···	$\mathbf{w}_M$

上面的表格说明了我们希望将实际排名情况R分解成两个矩阵（V和W）的乘积形式。V的维度是$\breve{d}$xN的，N是用户个数，$\breve{d}$可以是影片类型，例如（喜剧片，爱情片，悬疑片，动作片，...）。根据用户喜欢的类型不同，赋予不同的权重。W的维度是$\breve{d}$xM，M是电影数目，$\breve{d}$同样是影片类型，该部电影属于哪一类型就在那个类型上占比较大的权重。当然，$\breve{d}$维特征不一定就是影片类型，还可以是其它特征，例如明显阵容、年代等等。





Matrix Factorization Model

learning:

known rating

→ learned **factors** $\mathbf{v}_n$ and $\mathbf{w}_m$

→ unknown rating prediction

similar modeling can be used for other **abstract features**

那么，Matrix Factorization的目标就是最小化E_{in}函数。E_{in}表达式如下所示：

$$\min_{\mathbf{W},\mathbf{V}} E_{\text{in}}(\{\mathbf{w}_m\},\{\mathbf{v}_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} \left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right)^2$$

$$= \sum_{m=1}^{M} \left(\sum_{(\mathbf{x}_n, r_{nm}) \in \mathcal{D}_m} \left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right)^2 \right)$$

E_{in}中包含了两组待优化的参数，分别是v_n和w_m。我们可以借鉴上节课中k-Means的做法，将其中第一个参数固定，优化第二个参数，然后再固定第二个参数，优化第一个参数，一步一步进行优化。

当v_n固定的时候，只需要对每部电影做linear regression即可，优化得到每部电影的$\check{d}$维特征值w_m。

当w_m固定的时候，因为V和W结构上是对称的，同样只需要对每个用户做linear regression即可，优化得到每个用户对$\check{d}$维电影特征的喜爱程度v_n。

- two sets of variables:
  can consider **alternating minimization, remember? :-)**
- when $\mathbf{v}_n$ fixed, minimizing $\mathbf{w}_m \equiv$ minimize E_{in} within $\mathcal{D}_m$
  —simply per-movie (per-$\mathcal{D}_m$) **linear regression** without w_0
- when $\mathbf{w}_m$ fixed, minimizing $\mathbf{v}_n$?
  —per-user linear regression without v_0

by **symmetry** between users/movies



这种算法叫做alternating least squares algorithm。它的处理思想与k-Means算法相同，其算法流程图如下所示：

Alternating Least Squares

1. initialize $\tilde{d}$ dimension vectors $\{\mathbf{w}_m\}, \{\mathbf{v}_n\}$
2. alternating optimization of E_{in}: repeatedly
   1. optimize $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_M$: update $\mathbf{w}_m$ by m-th-movie linear regression on $\{(\mathbf{v}_n, r_{nm})\}$
   2. optimize $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_N$: update $\mathbf{v}_n$ by n-th-user linear regression on $\{(\mathbf{w}_m, r_{nm})\}$

   until converge

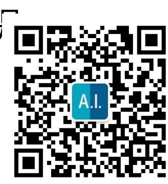
alternating least squares algorithm有两点需要注意。第一是initialize问题，通常会随机选取v_n和w_m。第二是converge问题，由于每次迭代更新都能减小E_{in}，E_{in}会趋向于0，则保证了算法的收敛性。

- initialize: usually just randomly
- converge: guaranteed as E_{in} decreases during alternating minimization

在上面的分析中，我们提过Matrix Factorization与Linear Autoencoder的相似性，下图列出了二者之间的比较。

Linear Autoencoder	Matrix Factorization
$\mathrm{X} \approx \mathrm{W}(\mathrm{W}^T\mathrm{X})$	$\mathrm{R} \approx \mathrm{V}^T\mathrm{W}$
motivation: special d-$\tilde{d}$-d linear NNet	motivation: N-$\tilde{d}$-M linear NNet
error measure: squared on all x_{ni}	error measure: squared on known r_{nm}
solution: global optimal at eigenvectors of $\mathrm{X}^T\mathrm{X}$	solution: local optimal via alternating least squares
usefulness: extract dimension-reduced features	usefulness: extract hidden user/movie features

Matrix Factorization与Linear Autoencoder有很强的相似性，都可以从原始资料汇总提取有用的特征。其实，linear autoencoder可以看成是matrix factorization的一种特殊



形式。

Stochastic Gradient Descent

我们刚刚介绍了alternating least squares algorithm来解决Matrix Factorization的问题。这部分我们将讨论使用Stochastic Gradient Descent方法来进行求解。之前的alternating least squares algorithm中，我们考虑了所有用户、所有电影。现在使用SGD，随机选取一笔资料，然后只在与这笔资料有关的error function上使用梯度下降算法。使用SGD的好处是每次迭代只要处理一笔资料，效率很高；而且程序简单，容易实现；最后，很容易扩展到其它的error function来实现。

$$E_{\text{in}}(\{\mathbf{w}_m\},\{\mathbf{v}_n\}) \propto \sum_{\text{user } n \text{ rated movie } m} \underbrace{\left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right)^2}_{\text{err(user } n,\ \text{movie } m,\ \text{rating } r_{nm})}$$

SGD: randomly pick **one example** within the $\sum$ &
update with **gradient to per-example** err, **remember? :-)**

- **'efficient'** per iteration
- **simple** to implement
- easily extends to **other** err

对于每笔资料，它的error function可表示为：

$$\text{err}(\text{user } n,\ \text{movie } m,\ \text{rating } r_{nm}) = \left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right)^2$$

上式中的err是squared error function，仅与第n个用户v_n，第m部电影w_m有关。其对v_n和w_m的偏微分结果为：

$$\nabla v_n = -2(r_{nm} - w_m^T v_n) w_m$$

$$\nabla w_m = -2(r_{nm} - w_m^T v_n) v_n$$



$$\nabla_{\mathbf{v}_{1126}} \quad \text{err}(\text{user } n, \text{ movie } m, \text{rating } r_{nm}) = \mathbf{0} \text{ unless } n = 1126$$

$$\nabla_{\mathbf{w}_{6211}} \quad \text{err}(\text{user } n, \text{ movie } m, \text{rating } r_{nm}) = \mathbf{0} \text{ unless } m = 6211$$

$$\nabla_{\mathbf{v}_n} \quad \text{err}(\text{user } n, \text{ movie } m, \text{rating } r_{nm}) = -2\left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right) \mathbf{w}_m$$

$$\nabla_{\mathbf{w}_m} \quad \text{err}(\text{user } n, \text{ movie } m, \text{rating } r_{nm}) = -2\left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right) \mathbf{v}_n$$

很明显，∇v_n和∇w_m都由两项乘积构成。（忽略常数因子2）。第一项都是$r_{nm} - w_m^T v_n$，即余数residual。我们在之前介绍的GBDT算法中也介绍过余数这个概念。∇v_n的第二项是w_m，而∇w_m的第二项是v_n。二者在结构上是对称的。

计算完任意一个样本点的SGD后，就可以构建Matrix Factorization的算法流程。SGD for Matrix Factorization的算法流程如下所示：

SGD for Matrix Factorization

initialize $\tilde{d}$ dimension vectors $\{\mathbf{w}_m\}, \{\mathbf{v}_n\}$ **randomly**

for $t = 0, 1, \ldots, T$

1. randomly pick (n, m) within all known r_{nm}
2. calculate residual $\tilde{r}_{nm} = \left(r_{nm} - \mathbf{w}_m^T \mathbf{v}_n\right)$
3. SGD-update:

$$\mathbf{v}_n^{new} \leftarrow \mathbf{v}_n^{old} + \eta \cdot \tilde{r}_{nm} \mathbf{w}_m^{old}$$

$$\mathbf{w}_m^{new} \leftarrow \mathbf{w}_m^{old} + \eta \cdot \tilde{r}_{nm} \mathbf{v}_n^{old}$$

在实际应用中，由于SGD算法简单高效，Matrix Factorization大多采用这种算法。

介绍完SGD for Matrix Factorization之后，我们来看一个实际的应用例子。问题大致是这样的：根据现在有的样本资料，预测未来的趋势和结果。显然，这是一个与时间先后有关的预测模型。比如说一个用户三年前喜欢的电影可能现在就不喜欢了。所以在使用SGD选取样本点的时候有一个技巧，就是最后T次迭代，尽量选择时间上靠后的样本放入到SGD算法中。这样最后的模型受这些时间上靠后的样本点影响比较大，也相对来说比较准确，对未来的预测会比较准。



KDDCup 2011 Track 1: World Champion Solution by NTU

- specialty of data (application need):
  per-user training ratings **earlier than** test ratings **in time**
- training/test mismatch: typical **sampling bias, remember? :-)**

- want: **emphasize latter** examples
- **last** T' iterations of SGD: **only those** T' **examples** considered
  —learned $\{\mathbf{w}_m\}, \{\mathbf{v}_n\}$ **favoring** those
- our idea: **time-deterministic** ~~S~~GD that visits **latter** examples **last**
  —**consistent improvements** of test performance

所以，在实际应用中，我们除了使用常规的机器学习算法外，还需要根据样本数据和问题的实际情况来修改我们的算法，让模型更加切合实际，更加准确。我们要学会灵活运用各种机器学习算法，而不能只是照搬。

Summary of Extraction Models

从第12节课开始到现在，我们总共用了四节课的时间来介绍Extraction Models。虽然我们没有给出Extraction Models明确的定义，但是它主要的功能就是特征提取和特征转换，将原始数据更好地用隐藏层的一些节点表征出来，最后使用线性模型将所有节点aggregation。这种方法使我们能够更清晰地抓住数据的本质，从而建立最佳的机器学习模型。

下图所示的就是我们介绍过的所有Extraction Models，除了这四节课讲的内容之外，还包括之前介绍的Adaptive/Gradient Boosting模型。因为之前笔记中都详细介绍过，这里就不再一一总结了。



Adaptive/Gradient Boosting
hypotheses g_t; weights α_t

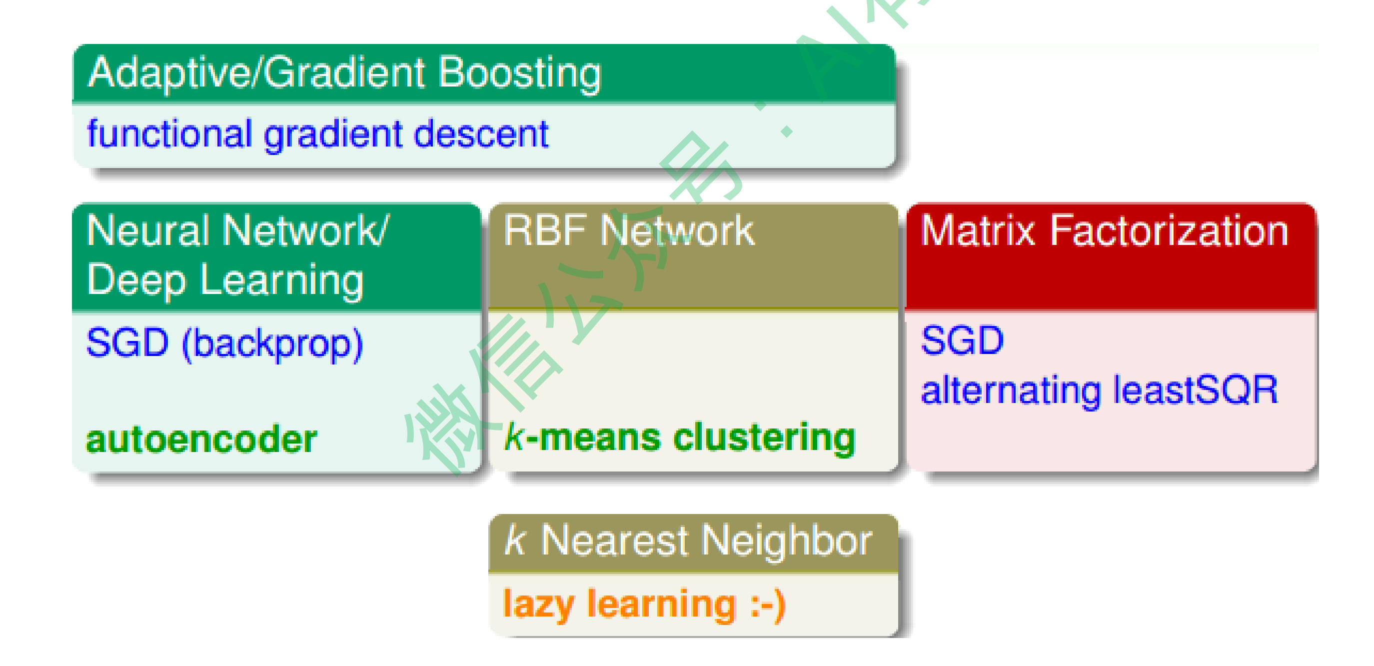
Neural Network/ Deep Learning
weights $w_{ij}^{(\ell)}$;
weights $w_{ij}^{(L)}$

RBF Network
RBF centers $\boldsymbol{\mu}_m$;
weights β_m

Matrix Factorization
user features $\mathbf{v}_n$;
movie features $\mathbf{w}_m$

k Nearest Neighbor
$\mathbf{x}_n$-neighbor RBF;
weights y_n

除了各种Extraction Models之外，我们这四节课还介绍了不同的Extraction Techniques。下图所示的是对应于不同的Extraction Models的Extraction Techniques。



最后，总结一下这些Extraction Models有什么样的优点和缺点。从优点上来说：

- **easy：机器自己完成特征提取，减少人类工作量**
- **powerful：能够处理非常复杂的问题和特征提取**

另一方面，从缺点上来说：

- **hard：通常遇到non-convex的优化问题，求解较困难，容易得到局部最优解而非全局最优解**
- **overfitting：模型复杂，容易造成过拟合，需要进行正则化处理**



所以说，Extraction Models是一个非常强大的机器学习工具，但是使用的时候也要小心处理各种可能存在的问题。

Neural Network/ Deep Learning | RBF Network | Matrix Factorization

Pros

- 'easy': reduces human burden in designing features
- powerful: if enough hidden variables considered

Cons

- 'hard': non-convex optimization problems in general
- overfitting: needs proper regularization/validation

be careful when applying extraction models

总结

本节课主要介绍了Matrix Factorization。从电影推荐系统模型出发，首先，我们介绍了Linear Network。它从用户ID编码后的向量中提取出有用的特征，这是典型的feature extraction。然后，我们介绍了基本的Matrix Factorization算法，即alternating least squares，不断地在用户和电影之间交互地做linear regression进行优化。为了简化计算，提高运算速度，也可以使用SGD来实现。事实证明，SGD更加高效和简单。同时，我们可以根据具体的问题和需求，对固有算法进行一些简单的调整，来获得更好的效果。最后，我们对已经介绍的所有Extraction Models做个简单的总结。Extraction Models在实际应用中是个非常强大的工具，但是也要避免出现过拟合等问题。

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程



林轩田《机器学习技法》课程笔记16（完结） -- Finale

作者：红色石头　　公众号：AI有道（id：redstonewill）

上节课我们主要介绍了Matrix Factorization。通过电影推荐系统的例子，介绍Matrix Factorization其实是一个提取用户特征，关于电影的线性模型。反过来也可以看出是关于用户的线性模型。然后，我们使用SGD对模型进行最佳化。本节课我们将对机器学习技法课程介绍过的所有内容做个总结，分成三个部分：Feature Exploitation Techniques，Error Optimization Techniques和Overfitting Elimination Techniques。

Feature Exploitation Techniques

我们在本系列课程中介绍的第一个特征提取的方法就是kernel。Kernel运算将特征转换和计算内积这两个步骤合二为一，提高了计算效率。我们介绍过的kernel有：Polynormial Kernel、Gaussian Kernel、Stump Kernel等。另外，我们可以将不同的kernels相加（transform union）或者相乘（transform combination），得到不同的kernels的结合形式，让模型更加复杂。值得一提的是，要成为kernel，必须满足Mercer Condition。不同的kernel可以搭配不同的kernel模型，比如：SVM、SVR和probabilistic SVM等，还包括一些不太常用的模型：kernel ridge regression、kernel logistic regression。使用这些kernel模型就可以将线性模型扩展到非线性模型，kernel就是实现一种特征转换，从而能够处理非常复杂的非线性模型。顺便提一下，因为PCA、k-Means等算法都包含了内积运算，所以它们都对应有相应的kernel版本。



Exploiting Numerous Features via Kernel

numerous features within some Φ:
embedded in kernel K_Φ with inner product operation

Polynomial Kernel	Gaussian Kernel	Stump Kernel
'scaled' polynomial transforms	infinite-dimensional transforms	decision-stumps as transforms

Sum of Kernels	Product of Kernels	Mercer Kernels
transform union	transform combination	transform implicitly

	kernel ridge regression	kernel logistic regression
SVM	SVR	probabilistic SVM

possibly **Kernel PCA**, **Kernel k-Means**, . . .

Kernel是我们利用特征转换的第一种方法，那利用特征转换的第二种方法就是Aggregation。我们之前介绍的所有的hypothesis都可以看成是一种特征转换，然后再由这些g组合成G。我们介绍过的分类模型（hypothesis）包括：Decision Stump、Decision Tree和Gaussian RBF等。如果所有的g是已知的，就可以进行blending，例如Uniform、Non-Uniform和Conditional等方式进行aggregation。如果所有的g是未知的，可以使用例如Bagging、AdaBoost和Decision Tree的方法来建立模型。除此之外，还有probabilistic SVM模型。值得一提的是，机器学习中很多模型都是类似的，我们在设计一个机器学习模型时，应该融会贯通。



Exploiting Predictive Features via Aggregation

predictive features within some $\mathbf{\Phi}$:

$$\phi_t(\mathbf{x}) = g_t(\mathbf{x})$$

Decision Stump	Decision Tree	(Gaussian) RBF
simplest perceptron; simplest DecTree	branching (divide) + leaves (conquer)	prototype (center) + influence
Uniform	**Non-Uniform**	**Conditional**
Bagging; Random Forest	AdaBoost; GradientBoost	Decision Tree; Nearest Neighbor
	probabilistic SVM	

possibly **Infinite Ensemble Learning**, **Decision Tree SVM**, ...

除此之外，我们还介绍了利用提取的方式，找出潜藏的特征（Hidden Features）。一般通过unsupervised learning的方法，从原始数据中提取出隐藏特征，使用权重表征。相应的模型包括：Neural Network、RBF Network、Matrix Factorization等。这些模型使用的unsupervised learning方法包括：AdaBoost、k-Means和Autoencoder、PCA等。



Exploiting Hidden Features via Extraction

hidden features within some **Φ**:

as hidden variables to be 'jointly' optimized with usual weights

—possibly with the help of **unsupervised learning**

Neural Network; Deep Learning	RBF Network	Matrix Factorization
neuron weights	RBF centers	user/movie factors

AdaBoost; GradientBoost	*k*-Means	Autoencoder; PCA
g_t parameters	cluster centers	'basis' directions

possibly **GradientBoosted Neurons**,
NNet on Factorized Features, . . .

另外，还有一种非常有用的特征转换方法是维度压缩，即将高维度的数据降低（投影）到低维度的数据。我们介绍过的维度压缩模型包括：Decision Stump、Random Forest Tree Branching、Autoencoder、PCA和Matrix Factorization等。这些从高纬度到低纬度的特征转换在实际应用中作用很大。



Exploiting Low-Dim. Features via Compression

low-dimensional features within some Φ:

compressed from original features

Autoencoder;PCA
info.-preserving compression

Decision Stump; DecTree Branching
'best' naïve projection to $\mathbb{R}$

Random Forest Tree Branching
'random' low-dim. projection

Matrix Factorization
projection from abstract to concrete

Feature Selection
'most-helpful' low-dimensional projection

possibly other 'dimension reduction' models

Error Optimization Techniques

接下来我们将总结一下本系列课程中介绍过哪些优化技巧。首先，第一个数值优化技巧就是梯度下降（Gradient Descent），即让变量沿着其梯度反方向变化，不断接近最优解。例如我们介绍过的SGD、Steepest Descent和Functional GD都是利用了梯度下降的技巧。



Numerical Optimization via Gradient Descent

when ∇E 'approximately' defined, use it for **1st order approximation**:

$$\text{new variables} = \text{old variables} - \eta \nabla E$$

SGD/Minibatch/GD	Steepest Descent	Functional GD
(Kernel) LogReg; Neural Network [backprop]; Matrix Factorization; Linear SVM (maybe)	AdaBoost; GradientBoost	AdaBoost; GradientBoost

possibly **2nd order techniques**, **GD under constraints**, . . .

而对于一些更复杂的最佳化问题，无法直接利用梯度下降方法来做，往往需要一些数学上的推导来得到最优解。最典型的例子是Dual SVM，还包括Kernel LogReg、Kernel RidgeReg和PCA等等。这些模型本身包含了很多数学上的一些知识，例如线性代数等等。除此之外，还有一些boosting和kernel模型，虽然本课程中没有提到，但是都会用到类似的数学推导和转换技巧。

Indirect Optimization via Equivalent Solution

when difficult to solve original problem, seek for **equivalent solution**

Dual SVM	Kernel LogReg Kernel RidgeReg	PCA
equivalence via convex QP	equivalence via representer	equivalence to eigenproblem

some **other boosting models** and **modern solvers of kernel models** rely on such a technique heavily

如果原始问题比较复杂，求解比较困难，我们可以将原始问题拆分为子问题以简化计



算。也就是将问题划分为多个步骤进行求解，即Multi-Stage。例如probabilistic SVM、linear blending、RBF Network等。还可以使用交叉迭代优化的方法，即Alternating Optim。例如k-Means、alternating LeastSqr等。除此之外，还可以采样分而治之的方法，即Divide & Conquer。例如decision tree。

Complicated Optimization via Multiple Steps

when difficult to solve original problem,
seek for 'easier' sub-problems

Multi-Stage	Alternating Optim.	Divide & Conquer
probabilistic SVM;	*k*-Means;	decision tree;
linear blending;	alternating LeastSqr;	
stacking;	(steepest descent)	
RBF Network;		
DeepNet pre-training		

useful for complicated models

Overfitting Elimination Techniques

Feature Exploitation Techniques和Error Optimization Techniques都是为了优化复杂模型，减小E_{in}。但是E_{in}太小有很可能会造成过拟合overfitting。因此，机器学习中，Overfitting Elimination尤为重要。

首先，可以使用Regularization来避免过拟合现象发生。我们介绍过的方法包括：large-margin、L2、voting/averaging等等。



Overfitting Elimination via Regularization

when model too 'powerful':

add brakes somewhere

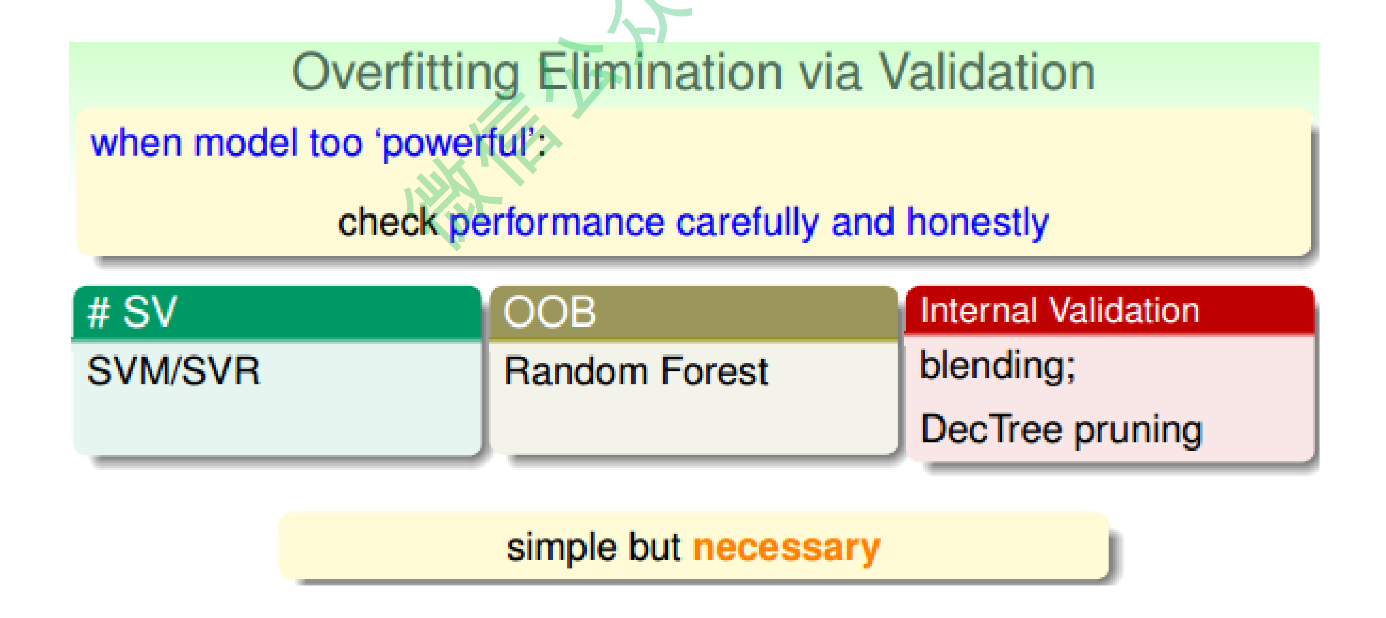
large-margin	L2	voting/averaging
SVM;	SVR;	uniform blending;
AdaBoost (indirectly)	kernel models;	Bagging;
	NNet [weight-decay]	Random Forest

denoising	weight-elimination	constraining
autoencoder	NNet	autoenc. [weights];
		RBF [# centers];

pruning	early stopping
decision tree	NNet (any GD-like)

arguably most important techniques

除了Regularization之外，还可以使用Validation来消除Overfitting。我们介绍过的Validation包括：SV、OOB和Internal Validation等。



Machine Learning in Action

本小节介绍了林轩田老师所在的台大团队在近几年的KDDCup国际竞赛上的表现和使用的各种机器算法。融合了我们在本系列课程中所介绍的很多机器学习技法和模型。这里不再一一赘述，将相应的图片贴出来，读者自己看看吧。



NTU KDDCup 2010 World Champion Model

Feature engineering and classifier ensemble for KDD Cup 2010,
Yu et al., KDDCup 2010

linear blending of

Logistic Regression +
many rawly encoded features

Random Forest +
human-designed features

yes, you've learned everything! :-)

NTU KDDCup 2011 Track 1 World Champion Model

A linear ensemble of individual and blended models for music rating prediction,
Chen et al., KDDCup 2011

NNet, DecTree-like, and then linear blending of

- Matrix Factorization variants, including probabilistic PCA
- Restricted Boltzmann Machines: an 'extended' autoencoder
- *k* Nearest Neighbors
- Probabilistic Latent Semantic Analysis:
  an extraction model that has 'soft clusters' as hidden variables
- linear regression, NNet, & GBDT

**yes, you can 'easily'
understand everything! :-)**



NTU KDDCup 2012 Track 2 World Champion Model

A two-stage ensemble of diverse models for advertisement ranking in KDD Cup 2012, Wu et al., KDDCup 2012

NNet, GBDT-like, and then linear blending of

- Linear Regression variants, including linear SVR
- Logistic Regression variants
- Matrix Factorization variants
- ...

'key' is to **blend properly without overfitting**

NTU KDDCup 2013 Track 1 World Champion Model

Combination of feature engineering and ranking models for paper-author identification in KDD Cup 2013, Li et al., KDDCup 2013

linear blending of

- Random Forest with many many many trees
- GBDT variants

with tons of efforts in designing features

'another key' is to **construct features with domain knowledge**

ICDM在2006年的时候发布了排名前十的数据挖掘算法，如下图所示。其中大部分的算法我们在本系列的课程中都有过介绍。值得一提的是Naive Bayes算法本课程中没有涉及，贝叶斯模型在实际中应用还是挺广泛的，后续可能还需要深入学习一下。



ICDM 2006 Top 10 Data Mining Algorithms

1. C4.5: another **decision tree**
2. *k*-Means
3. SVM
4. Apriori: for frequent itemset mining
5. EM: **'alternating optimization'** algorithm for some models
6. PageRank: for link-analysis, similar to **matrix factorization**
7. AdaBoost
8. *k* Nearest Neighbor
9. Naive Bayes: a simple **linear model** with 'weights' decided by data statistics
10. C&RT

personal view of five missing ML competitors:
LinReg, LogReg,
Random Forest, GBDT, NNet

最后，我们将所有介绍过的机器学习算法和模型列举出来：

Machine Learning Jungle

bagging　decision tree　support vector machine　**neural network**　*kernel*
AdaBoost　aggregation　*sparsity*　autoencoder　**functional gradient**
dual　uniform blending　deep learning　nearest neighbor　decision stump
kernel LogReg　large-margin　*prototype*　quadratic programming　**SVR**
GBDT　**PCA**　random forest　*matrix factorization*　**Gaussian kernel**
soft-margin　*k-means*　OOB error　**RBF network**　probabilistic SVM

welcome to the **jungle**!

总结

本节课主要从三个方面来对机器学习技法课程做个总结：Feature Exploitation Techniques，Error Optimization Techniques和Overfitting Elimination Techniques。最后介绍了林轩田老师带领的台大团队是如何在历届KDDCup中将很多机器学习算法模



型融合起来，并获得了良好的成绩。

- Feature Exploitation Techniques
  kernel, aggregation, extraction, low-dimensional
- Error Optimization Techniques
  gradient, equivalence, stages
- Overfitting Elimination Techniques
  (lots of) regularization, validation
- Machine Learning in Practice
  welcome to the jungle

注明：

文章中所有的图片均来自台湾大学林轩田《机器学习技法》课程



